NEW PROBLEMS IN EXPLORING DISTRIBUTED DATA

by

Mingwang Tang

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

May 2015

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Mingwang Tang**

has been approved by the following supervisory committee members:

Feifei Li	, Chair	**10/02/14** Date Approved
Jeffrey M. Phillips	, Member	**10/02/14** Date Approved
Suresh Venkatasubramanian	, Member	**10/02/14** Date Approved
Mohamed F. Mokbel	, Member	**10/02/14** Date Approved
Tingjian Ge	, Member	**10/02/14** Date Approved

and by **Ross Whitaker**, Chair/Dean of

the Department/College/School of **Computing**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

In the era of big data, many applications generate continuous online data from distributed locations, scattering devices, etc. Examples include data from social media, financial services, and sensor networks, etc. Meanwhile, large volumes of data can be archived or stored offline in distributed locations for further data analysis. Challenges from data uncertainty, large-scale data size, and distributed data sources motivate us to revisit several classic problems for both online and offline data explorations.

The problem of continuous threshold monitoring for distributed data is commonly encountered in many real-world applications. We study this problem for distributed probabilistic data. We show how to prune expensive threshold queries using various tail bounds and combine tail-bound techniques with adaptive algorithms for monitoring distributed deterministic data. We also show how to approximate threshold queries based on sampling techniques.

Threshold monitoring problems can only tell a monitoring function is above or below a threshold constraint but not how far away from it. This motivates us to study the problem of continuous tracking functions over distributed data. We first investigate the tracking problem on a chain topology. Then we show how to solve tracking problems on a distributed setting using solutions for the chain model. We studied online tracking of the max function on "broom" tree and general tree topologies in this work.

Finally, we examine building scalable histograms for distributed probabilistic data. We show how to build approximate histograms based on a partition-and-merge principle on a centralized machine. Then, we show how to extend our solutions to distributed and parallel settings to further mitigate scalability bottlenecks and deal with distributed data.

I dedicate this thesis to my family, who gives me all-enduring and selfless love on my way through PhD study.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

More than five years has passed since the first day I began my PhD study in the US. I am very lucky to experience two kinds of academic style from both Florida State University (FSU) where I spent my first two years of PhD study and University of Utah where the end of my PhD journey will be. I would like to first thank to cs faculties in FSU, who help me to build a solid knowledge in core courses in computer science.

I would like to express my gratitude to my advisor Feifei Li, who gives me a great guidance on my research study. From our academic discussions, I have learned so much toward doing research, for instance, how to mathematically formalize a problem definition and how to approach the problem from different views and others. He is a very forthright person and will point out my shortcomings to help me to realize the places where I need to improve. I would never be able to reach the closing of this journey without his guidance, advices and even criticisms. I would like to thank Feifei for being my advisor and supporting my PhD studies. Also, I would like to thank Prof. Jeff Phillips, who broadened my views and research skills from our collaborations. I thank Prof. Suresh Venkatasubramanian for his insightful comments for my presentations in our data group meeting. I also want to thank Prof. Mohamed F. Mokbel and Prof. Tingjian Ge for serving on my supervisory committee.

Finally, I thank NSF for funding me for my PhD research projects.

CHAPTER 1

INTRODUCTION

The goal of this dissertation research is to design, implement, and evaluate novel data exploration techniques on distributed data to support scalable data analytics and decision-making systems. Therefore, we have studied three closely related problems in this thesis: (I) how to continuously monitor distributed probabilistic data against a probabilistic threshold; (II) how to continuously track function changes over distributed data; (III) how to build scalable histograms on large probabilistic data.

1.1 Motivation and Background

Data exploration tasks aim to retrieve relevant information from a large dataset, which becomes more challenging in the era of big data. In many applications, large volumes of data are generated from scattering devices or distributed sources continuously. Examples include data collected from network intrusion detection systems, measurements from large sensor networks, and locations and application data from location-based services, as shown in Figure 1.1.

Therefore, exploring distributed data in an online fashion becomes critical for further data analytics and decision-making applications. A natural example is continuous distributed monitoring, where a function of distributed observations is continuously moni-

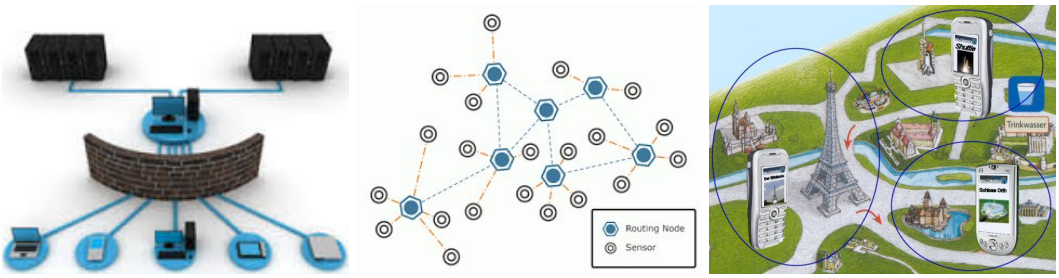


Figure 1.1: Distributed streaming data in applications

tored against a user-specified threshold and an alarm will be generated when the threshold constraint is violated.

In the distributed setting, it is always desirable, sometimes even critical, to reduce the communication cost for a number of reasons [3, 15–17, 26, 55, 56, 63, 86]. For instance, in sensor networks, transmitting messages consumes valuable on-board battery resources of sensor nodes, which in fact determines the lifetime of these networks. From the network infrastructure's point of view (e.g., ISP), transmitting the monitoring data to perform the distributed computation is impractical, which will seriously affect the network bandwidth.

An emerging challenge for monitoring distributed data is uncertainty, which is inherently introduced when massive amounts of data are generated in distributed sources. For instance, measurements in a sensor network are inherently noisy; data integration systems produce fuzzy matches. A motivating example in our thesis is the Shipboard Automated Meteorological and Oceanographic System (SAMOS) project [70].

We have observed that in SAMOS: 1) meteorological data are observed from research vessels, ships, and towers which are naturally distributed; 2) ambiguity, errors, imprecise readings, and uncertainty are present in the real-time data collected, due to hazardous conditions, coarse real-time measurement, and multiple readings for the same observation; 3) large amounts of data (e.g., wind speed, temperature, humidity, etc.) need to be processed in less than a minute continuously.

It is useful to represent data in SAMOS (as well as other applications previously discussed) using *distributed probabilistic data*. For instance, a common practice in SAMOS is for each ship/tower to buffer data for each interval (e.g., 5 minutes) and send one representative for data in an interval. Clearly, modeling data in a given interval using probabilistic data, such as a probability distribution function (pdf), is no doubt a viable and attractive solution (especially when we want to also account for the presence of uncertainty and errors

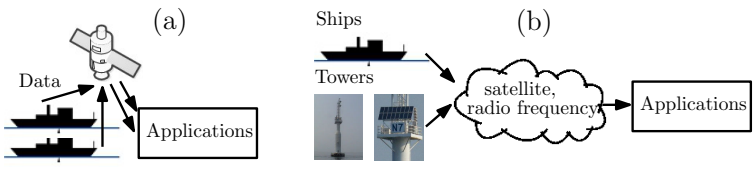


Figure 1.2: Shipboard Automated Meteorological and Oceanographic System.

in the raw readings).

Existing techniques of threshold monitoring on deterministic distributed data [18, 48, 52, 74] cannot be directly applied to distributed probabilistic data as in the SAMOS system. This motivates us to investigate a novel threshold monitoring problem on distributed probabilistic data. We extend threshold queries to probabilistic data by considering a probabilistic-threshold semantics [10, 22, 77]. The goal of this part of our dissertation is to efficiently explore distributed data with uncertainty and produce an alarm when a true threshold crossing has happened with a high probability. One design principle we have applied in tackling computation and communication cost on distributed probabilistic data is utilizing tail-bound filters in the monitoring instance. Then, techniques of threshold monitoring for distributed deterministic data could be used to further reduce communication cost.

Online threshold monitoring can only tell whether the observing function of distributed data are above or below a threshold but not how far it is from the threshold. It is useful to continuously tracking various functions of distributed data. For instance, SAMOS users may be interested in continuously tracking the maximum of current temperature readings for a number of ships from a region in proximity. Such examples can be easily found in location-based services. Recent studies [87, 89] focused on the online tracking problem with only one observer and one tracker, which has limited application to multiple observers in distributed locations. To this end, we have studied the problem of tracking functions of distributed data (each data source can be described by an arbitrary function) continuously in an online fashion. To achieve 100% accuracy for continuous online tracking of arbitrary functions, any change of an observing function will lead to a message to the tracker, which simply generates excessive communications. Similar to the strategies in [87, 89], we relax the requirement by allowing a maximum error of Δ for the function under tracking by the tracker. We first investigated the tracking problem on a chain topology, where the observer connects to the tracker through multiple relay nodes. Following that, we studied the tracking problem where distributed observers connect to the tracker through a "broom" tree model and a general-tree topology.

At the back end, distributed data often converge in a data warehouse, where in-depth data explorations and analysis are possible. The challenge is large data size and uncertainty

caused by data integration or distributed data sources, as we see in SAMOS data. Data summary techniques, e.g., histograms, provide an effective venue for exploring large probabilistic data. Recently, histograms on probabilistic data have been proposed to work with probabilistic datasets [12–14]. However, existing studies suffer from limited scalability and do not adapt to large-scale data size and distributed data sources. This motivates us to study scalable histograms on large probabilistic data. We address this problem by building scalable histograms on large probabilistic data using a partition and merge principle. We also extend our solutions to distributed and parallel settings to mitigate scalability bottlenecks and deal with distributed data.

1.2 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapter 2, we study the problem of threshold monitoring for distributed probabilistic data. One outstanding challenge is that answering queries on probabilistic data is in #P-complete complexity. We first leveraged tail-bound techniques to help assert the threshold violations and combined them with techniques for threshold monitoring on distributed deterministic data to further reduce communication cost. When the tail bounds fail to make a decision, we propose effective sampling methods to approximate threshold queries. In Chapter 3, we study the problem of distributed online tracking. We first investigated optimal tracking on chain topologies, which gives us more insights for distributed online tracking on a general-tree topology. With large probabilistic data, building a succinct synopsis, e.g., histograms, becomes meaningful for data exploration and data management tasks. We study the scalable histogram construction on probabilistic data in Chapter 4 in an effort to support large-scale datasets and distributed and parallel processing on large probabilistic data. Finally, we conclude and discuss some open problems in Chapter 6.

CHAPTER 2

THRESHOLD MONITORING FOR DISTRIBUTED PROBABILISTIC DATA

2.1 Introduction

When massive amounts of data are generated, uncertainty is inherently introduced at the same time. For instance, data integration produces fuzzy matches [28, 77]; in measurements, e.g., sensor readings, data are inherently noisy, and are better represented by a probability distribution rather than a single deterministic value [10, 25, 37, 77]. In a lot of these applications, data are generated at multiple sources, and collected from distributed networked locations. Examples include distributed observing stations, large sensor fields, geographically separate scientific institutes/units, and many more [54, 83]. A concrete example is the SAMOS project, as we mentioned in Chapter 1.

Meanwhile, as numerous studies in managing and exploring distributed data have shown, a primary concern is monitoring the distributed data and generating an alarm when a user-specified constraint is violated. A particular useful instance is the threshold-based constraint, which we refer to as the *distributed threshold monitoring* (DTM) problem and which has been extensively studied in distributed deterministic data [18, 48, 52, 74]. An application scenario is shown in Example 1.

Example 1. *Suppose each distributed site continuously captures temperature readings (one per system-defined time instance); the goal is to monitor them continuously and raise an alarm at the coordinator site whenever the* average temperature *from all sites* exceeds 80 *degrees at any time instance.*

Similar applications are required in exploring distributed probabilistic data. The notion of distributed threshold monitoring on probabilistic data is a critical problem, such as in the SAMOS system. The most natural and popular way of extending threshold queries to

probabilistic data is probabilistic-threshold semantics [10,22,77], which introduces another threshold on the probability of the query answer in addition to the threshold on the score value of the results. Consider the following example that extends Example 1:

Example 2. *Suppose readings in each site are now represented as probabilistic data (e.g., as we have just discussed for data in SAMOS), the goal is to monitor these readings continuously and raise an alarm at the coordinator site whenever* the probability *of the* average temperature *from all sites* exceeding 80 *degrees is* above 70% *at any time instance.*

We refer to them as the *distributed probabilistic threshold monitoring* (DPTM) problem. This variant is a robust alternative to DTM, more robust than the median, in that even if *all* sites report low-probability noise which skews their distributions, DPTM will only raise an alarm if a true threshold has been crossed, or what may have been noise occurs with high enough probability that it cannot be ignored. For the same reasons and motivations of its counterpart, the DTM problem, a paramount concern is to reduce the communication cost, measured by both the total number of messages and bytes communicated in the system. For example, on SAMOS, cutting down the communication cost would allow for the transmission of more accurate or diverse measurements. Due to the inherent difference in query processing between probabilistic and deterministic data, techniques developed from DTM are no longer directly applicable. This also brings up another challenge in DPTM, reducing the cpu cost, since query processing in probabilistic data is often computation-intensive, which is even worse in distributed probabilistic data [54].

We step up to these challenges and present a comprehensive study to the DPTM problem. Specifically:

- We formalize the DPTM problem in Section 2.2.
- We propose baseline methods in Section 2.3, which improve over the naive method of sending all tuples at each time instance.
- We design two efficient and effective monitoring methods in Section 2.4 that leverage moment generating functions and adaptive filters to significantly reduce the costs.
- When an exact solution is not absolutely necessary, we introduce novel sampling-based methods in Section 2.5 to further reduce the communication and the cpu costs.
- We extensively evaluate all proposed methods in Section 4.6 on large real data obtained from research vessels in the SAMOS project. The results have shown that our

monitoring methods have significantly outperformed the baseline approach. They also indicate that our sampling method is very effective when it is acceptable to occasionally miss one or two alarms with very small probability.

We discuss some useful extensions in Section 2.6, survey the related work in Section 4.7, and conclude in Section 4.8.

2.2 Problem Formulation

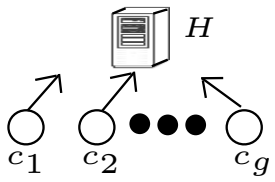
Sarma *et al.* [71] describe various models of uncertainty. We consider the attribute-level uncertain tuple that has been used frequently in the literature, and suits the applications for our problem well (e.g., data in SAMOS).

Each tuple has one or more uncertain attributes. Every uncertain attribute has a pdf for its value distribution. Correlation among attributes in one tuple can be represented by a joint pdf. This model has a lot of practical applications and is most suitable for measurements and readings [25, 50]. Without loss of generality, we assume that each tuple has only one uncertain attribute *score*. Let X_i be the random variable for the *score* of tuple d_i, where X_i can have either a discrete or a continuous pdf, with bounded size (see Figure 2.1(a)). Since each pdf is bounded, we assume that for all X_i's, $|X_i| \leq n$ for some value n where $|X_i|$ is the size of the pdf for X_i, which is the number of discrete values X_i may take for a discrete pdf, or the number of parameters describing X_i and its domain for a continuous pdf.

Given g distributed clients $\{c_1, \ldots, c_g\}$, and a centralized server H, we consider the *flat model* for the organization of distributed sites as shown in Figure 2.1(b); e.g., SAMOS uses the flat model. At each time instance t, for $t = 1, \ldots, T$, client c_i reports a tuple $d_{i,t}$ with a score $X_{i,t}$. We assume that data from different sites are independent. Similar assumptions were made in most distributed data monitoring or ranking studies [18, 39, 48, 52, 54, 62, 74].

tuples	attribute *score*
d_1	X_1
d_2	X_2
$\cdot$	$\cdots$
d_t	X_t

(a)



(b)

Figure 2.1: (a) Attribute-level uncertain tuple model. (b) The flat model.

Without loss of generality and for the ease of explanation, we assume that $X_{i,t} \in \mathbb{R}^+$. Our techniques can be easily extended to handle the case when $X_{i,t}$ may take negative values as well.

Definition 1 (DPTM)**.** *Given $\gamma \in \mathbb{R}^+$ and $\delta \in [0, 1)$, let $Y_t = \sum_{i=1}^{g} X_{i,t}$, for $t = 1, \ldots, T$. The goal is to raise an alarm at H, whenever for any $t \in [1, T]$ $\Pr[Y_t > \gamma] > \delta$.*

In our definition, DPTM monitors the *sum* constraint. Monitoring the *average* constraint is equivalent to this case, as well as any other types of constraints that can be expressed as a linear combination of one or more *sum* constraints.

As argued in Section 4.1, the goal is to minimize both the *overall* communication and computation costs, at the end of all time instances. We measure the communication cost using both the total number of bytes transmitted and the total number of messages sent. Lastly, when the context is clear, we omit the subscript t from Y_t and $X_{i,t}$.

2.3 Baseline Methods

At any time instance t, let $X_1, \ldots, X_g$ be the scores from c_1 to c_g and $Y = \sum_{i=1}^{g} X_i$. To monitor if $\Pr[Y > \gamma] > \delta$, the naive method is to ask each client c_i to send his score X_i to H, which is clearly very expensive.

2.3.1 Compute $\Pr[Y > \gamma]$ Exactly

The first challenge is to compute $\Pr[Y > \gamma]$ exactly at H. We differentiate two cases. When each X_i is represented by a discrete pdf, clearly, we can compute $Y_{1,2} = X_1 + X_2$ in $O(n^2)$ time by a nested loop over the possible values they may take. Next, we can compute $Y_{1,2,3} = X_1 + X_2 + X_3 = Y_{1,2} + X_3$ using $Y_{1,2}$ and X_3 in $O(n^3)$ time, since in the worst case the size of $Y_{1,2}$ is $O(n^2)$. We can recursively apply this idea to compute $Y = Y_{1,\ldots,g}$ in $O(n^g)$ time, then check $\Pr[Y > \gamma]$ exactly. But note that in this approach, since we did not sort the values in the output (to reduce the cost), in each step the discrete values in the output pdf is no longer sorted.

A better idea is to compute $Y_{1,\ldots,g/2}$, and $Y_{g/2+1,\ldots,g}$ separately, which only takes $O(n^{g/2})$ time. Then, by using the cdf (cumulative distribution function) of $Y_{g/2+1,\ldots,g}$, we can compute $\Pr[Y > \gamma]$ as follows:

$$\Pr[Y > \gamma] = \sum_{\forall y \in Y_{1,\ldots,g/2}} \Pr[Y_{1,\ldots,g/2} = y] \cdot \Pr[Y_{g/2+1,\ldots,g} > \gamma - y].$$

Computing the cdf of $Y_{g/2+1,\ldots,g}$ takes $O(n^{g/2}\log n^{g/2})$ in the worst case: as discussed above, discrete values in $Y_{g/2+1,\ldots,g}$ are not sorted. After which, finding out $\Pr[Y_{g/2+1,\ldots,g} > \gamma - y]$ for any y takes only constant time. Hence, this step takes $O(n^{g/2})$ time only (the size of $Y_{1,\ldots,g/2}$ in the worst case). So the overall cost of computing $\Pr[Y > \gamma]$ exactly at H becomes $O(n^{g/2}\log n^{g/2})$.

When some X_i's are represented by continuous pdfs, the above process no longer works. In this case, we leverage on the characteristic functions of X_i's to compute Y exactly. The characteristic function [6] of a random variable X is:

$$\varphi_X(\beta) = \mathrm{E}(e^{i\beta X}) = \int_{-\infty}^{+\infty} e^{i\beta x} f_X(x) d(x), \forall \beta \in \mathbb{R},$$

where i is the imaginary unit and $f_X(x)$ is the pdf of X. Let $\varphi_i(\beta)$ and $\varphi(\beta)$ be the characteristic functions of X_i and Y, respectively; a well-known result is that $\varphi(\beta) = \prod_{i=1}^{g} \varphi_i(\beta)$ [6]. Furthermore, by definition, $\varphi_i(\beta)$ and $\varphi(\beta)$ are the Fourier transform of the pdfs for X_i and Y, respectively. Hence, an immediate algorithm for computing the pdf of Y is to compute the Fourier transforms for the pdfs of X_i's, multiply them together to get $\varphi(\beta)$, then do an inverse Fourier transform on $\varphi(\beta)$ to obtain the pdf of Y. After which, we can easily check if $\Pr[Y > \gamma] > \delta$. The cost of this algorithm depends on the cost of each Fourier transform, which is dependent on the types of distributions being processed. Note that using this approach when all pdfs are discrete does not result in less running time than the method above: since the size of Y in the worst case is $O(n^g)$ (the pdf describing Y), this algorithm takes at least $O(n^g)$ time in the worst case, even though we can leverage on fast Fourier transform in this situation.

We denote the above algorithms as EXACTD and EXACTC, for the discrete and continuous cases, respectively.

2.3.2 Filtering by Markov Inequality

By the Markov inequality, we have $\Pr[Y > \gamma] \leq \frac{\mathrm{E}(Y)}{\gamma}$. Given that $\mathrm{E}(Y) = \sum_{i=1}^{g} \mathrm{E}(X_i)$, if each client X_i only sends $\mathrm{E}(X_i)$, H can check if $\frac{\mathrm{E}(Y)}{\gamma} < \delta$; if so, no alarm should be raised for sure; otherwise, we can then ask for X_i's, and apply the exact algorithm. We dub this approach the *Markov* method.

We can improve this further. Since $\mathrm{E}(Y) = \sum_{i=1}^{g} \mathrm{E}(X_i)$ and our goal is to monitor

if $\mathrm{E}(Y) < \gamma\delta$ by the Markov inequality, we can leverage on the adaptive thresholds algorithm for the DTM problem in deterministic data [48] to monitor if $\sum_{i=1}^{g} \mathrm{E}(X_i) < \gamma\delta$ continuously, which installs local filters at clients and adaptively adjusts them. Specifically, $\gamma\delta$ is treated as the global constraint; at each time instance, client c_i can compute $\mathrm{E}(X_i)$ locally which becomes a "deterministic score". Thus, the algorithm from [48] is applicable. Whenever it cannot assert an alarm at a time instance t, clients transmit X_i's to H and the server applies the exact algorithm (only for that instance). This helps reduce the communication cost and we dub this improvement the *Madaptive* method.

2.4 Improved Methods

We now improve on these baseline techniques. We replace the Markov Inequality through more complicated to apply, but more accurate, Chebyshev and Chernoff bounds (*Improved*). Then, we redesign *Improved* to leverage adaptive monitoring techniques designed for DTM (*Iadaptive*).

2.4.1 Improved Bounds on $\Pr[Y > \gamma]$

We first leverage on the general Chebyshev bound:

$$\Pr[|Y - \mathrm{E}(Y)| \geq a\sqrt{\mathrm{Var}(Y)}] \leq 1/a^2 \text{ for any } a \in \mathbb{R}^+,$$

which gives us the following one-sided forms:

$$\Pr[Y \geq \mathrm{E}(Y) + a] \leq \frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y) + a^2}, \forall a \in \mathbb{R}^+ \tag{2.1}$$

$$\Pr[Y \leq \mathrm{E}(Y) - a] \leq \frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y) + a^2}, \forall a \in \mathbb{R}^+. \tag{2.2}$$

When $\gamma > \mathrm{E}(Y)$, setting $a = \gamma - \mathrm{E}(Y)$ in (4.2) leads to:

$$\Pr[Y > \gamma] < \Pr[Y \geq \gamma] \leq \frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y) + (\gamma - \mathrm{E}(Y))^2}. \tag{2.3}$$

As such, when $\gamma > E(Y)$, if $\frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y)+(\gamma-\mathrm{E}(Y))^2} \leq \delta$, we definitely do not have to raise an alarm.

When $\gamma < \mathrm{E}(Y)$, we can set $a = \mathrm{E}(Y) - \gamma$ in (4.4) to get:

$$\Pr[Y \leq \gamma] \leq \frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y) + (\mathrm{E}(Y) - \gamma)^2}. \tag{2.4}$$

This implies that,

$$\Pr[Y > \gamma] = 1 - \Pr[Y \leq \gamma] > 1 - \frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y)+(\mathrm{E}(Y)-\gamma)^2}. \tag{2.5}$$

Hence, when $\gamma < \mathrm{E}(Y)$, as long as $1 - \frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y)+(\mathrm{E}(Y)-\gamma)^2} \geq \delta$, we should surely raise an alarm.

Given these observations, in each time instance, clients send $\mathrm{E}(X_i)$'s and $\mathrm{Var}(X_i)$'s to H, which computes $\mathrm{E}(Y)$ and $\mathrm{Var}(Y)$ locally (given that X_i's are independent from each other, $\mathrm{Var}(Y) = \sum_{i=1}^{g} \mathrm{Var}(X_i)$). Depending whether $\mathrm{E}(Y) > \gamma$ or $\mathrm{E}(Y) < \gamma$, H uses (4.5) or (2.5) to decide to raise or not to raise an alarm for this time instance. Nevertheless, this approach may still incur expensive communication and computation when $\mathrm{E}(Y) = \gamma$, or (4.5) ((2.5), resp.) does not hold when $\mathrm{E}(Y) > \gamma$ ($\mathrm{E}(Y) < \gamma$, resp.). It is also limited in the fact that H can only check either to raise an alarm or not to raise an alarm, but not both simultaneously, as $\mathrm{E}(Y) > \gamma$ and $\mathrm{E}(Y) < \gamma$ cannot hold at the same time.

We remedy these problems using the general Chernoff bound and the moment-generating function [6]. For any random variable Y, suppose its moment generating function is given by $M(\beta) = \mathrm{E}(e^{\beta Y})$ for any $\beta \in \mathbb{R}$, then:

$$\Pr[Y \geq a] \leq e^{-\beta a} M(\beta) \text{ for all } \beta > 0, \forall a \in \mathbb{R} \tag{2.6}$$

$$\Pr[Y \leq a] \leq e^{-\beta a} M(\beta) \text{ for all } \beta < 0, \forall a \in \mathbb{R} \tag{2.7}$$

Here, a can be any real value (positive or negative). Suppose the moment generating function of X_i and Y is $M_i(\beta)$ and $M(\beta)$, respectively, then $M(\beta) = \prod_{i=1}^{g} M_i(\beta)$ [6]. Hence, when the checking based on either (4.5) or (2.5) has failed, for any $\beta_1 > 0$ and $\beta_2 < 0$, the server requests c_i to calculate and send back $M_i(\beta_1)$ and $M_i(\beta_2)$. He computes $M(\beta_1)$ and $M(\beta_2)$, and by setting $a = \gamma$ in (2.6) and (4.7), he checks if:

$$\Pr[Y > \gamma] \leq \Pr[Y \geq \gamma] \leq e^{-\beta_1 \gamma} M(\beta_1) \leq \delta, \text{ and} \tag{2.8}$$

$$\Pr[Y > \gamma] = 1 - \Pr[Y \leq \gamma] > 1 - e^{-\beta_2 \gamma} M(\beta_2) \geq \delta. \tag{2.9}$$

When (2.8) holds, he does not raise an alarm; when (2.9) holds, he raises an alarm; only when both have failed, he requests X_i's for the exact computation.

Calculating $M_i(\beta)$ at a client c_i is easy. For a lot of parametric continuous pdfs, closed-

form expressions exist for their moment generating functions, or, one can use numeric methods to compute $M_i(\beta)$ to arbitrary precision for other continuous pdfs. For discrete pdfs, $M_i(\beta) = \sum_{x \in X_i} e^{\beta x} \Pr[X_i = x]$.

Another key issue is to figure out the optimal values for β_1 and β_2 in (2.8) and (2.9) to make the corresponding bound as tight as possible, which is to minimize $e^{-\beta_1 \gamma} M(\beta_1)$ and $e^{-\beta_2 \gamma} M(\beta_2)$ in (2.8) and (2.9), respectively. The *central limit theorem* states that the mean of a sufficiently large number of independent random variables will be approximately normally distributed, if each independent variable has finite mean and variance [6]. For a normal distribution with mean μ and variance σ^2, its moment generating function is $e^{\beta\mu + \frac{1}{2}\sigma^2\beta^2}$ for any $\beta \in \mathbb{R}$. Hence, let $Y' = \frac{1}{g}Y$, then Y' can be approximated by a normal distribution well, and we can approximate its moment generating function as:

$$M_{Y'}(\beta) \approx e^{\beta \mathrm{E}(Y') + \frac{1}{2}\mathrm{Var}(Y')\beta^2}, \forall \beta \in \mathbb{R}. \tag{2.10}$$

Note that $Y = gY'$, (2.8) and (2.10) imply that for any $\beta_1 > 0$:

$$\begin{aligned} \Pr[Y \geq \gamma] &= \Pr[Y' \geq \frac{\gamma}{g}] \leq e^{-\beta_1 \frac{\gamma}{g}} M_{Y'}(\beta_1) \\ &\approx e^{-\beta_1 \frac{\gamma}{g}} e^{\beta_1 \mathrm{E}(\frac{Y}{g}) + \frac{1}{2}\mathrm{Var}(\frac{Y}{g})\beta_1^2} \qquad \text{by (2.10)} \\ &= e^{\frac{\beta_1}{g}(\mathrm{E}(Y) - \gamma) + \frac{1}{2g^2}\mathrm{Var}(Y)\beta_1^2} \end{aligned} \tag{2.11}$$

Hence, we can approximate the optimal β_1 value for (2.8) by finding the β_1 value that minimizes the RHS of (2.11). Let $f(\beta_1)$ be the RHS of (2.11) and take its derivative w.r.t. β_1:

$$f'(\beta_1) = e^{\frac{\beta_1}{g}(\mathrm{E}(Y) - \gamma) + \frac{1}{2g^2}\mathrm{Var}(Y)\beta_1^2} \left(\frac{\mathrm{E}(Y) - \gamma}{g} + \frac{\mathrm{Var}(Y)}{g^2}\beta_1\right).$$

Let $f'(\beta_1) = 0$; we get $\beta_1 = \frac{g(\gamma - \mathrm{E}(Y))}{\mathrm{Var}(Y)}$. Furthermore, we can show that the second order derivative of $f(\beta_1)$,$f''(\beta_1)$, is always greater than 0 (we omit the details for brevity). That said, $f(\beta_1)$ (hence, the RHS of (2.11)) takes its minimal value when $\beta_1 = \frac{g(\gamma - \mathrm{E}(Y))}{\mathrm{Var}(Y)}$. Using a similar analysis, we can derive the optimal β_2 value. However, a constraint is that $\beta_1 > 0$ and $\beta_2 < 0$. That said, also with the observation that $f(\beta_1)$ (the corresponding function for β_2) is monotonically increasing when $\beta_1 > \frac{g(\gamma - \mathrm{E}(Y))}{\mathrm{Var}(Y)}$ ($\beta_2 < \frac{g(\gamma - \mathrm{E}(Y))}{\mathrm{Var}(Y)}$, respectively), let $\theta > 0$ be some small value,

$$\begin{cases} \beta_1 = \frac{g(\gamma - \mathrm{E}(Y))}{\mathrm{Var}(Y)}, \beta_2 = -\theta & \text{if } \gamma > \sum_{i=1}^{g} \mathrm{E}(X_i), \\ \beta_1 = \theta, \beta_2 = \frac{g(\gamma - \mathrm{E}(Y))}{\mathrm{Var}(Y)} & \text{if } \gamma < \sum_{i=1}^{g} \mathrm{E}(X_i), \\ \beta_1 = \theta, \beta_2 = -\theta & \text{otherwise,} \end{cases} \tag{2.12}$$

will help achieve tight bounds in (2.8) and (2.9).

This yields the *Improved* method, shown in Figure 2.2.

2.4.2 Improved Adaptive Threshold Monitoring

The *Improved* method needs at least g messages per time instance; to reduce this, we again leverage on the adaptive thresholds algorithm developed for work on DTM [48].

Consider (2.8) and (2.9), when we can continuously monitor if:

$$e^{-\beta_1 \gamma} \prod_{i=1}^{g} M(\beta_1) \leq \delta, \text{ or } 1 - e^{-\beta_2 \gamma} \prod_{i=1}^{g} M(\beta_2) \geq \delta \tag{2.13}$$

efficiently, whenever the first inequality in (2.13) holds at a time instance t, H knows for sure that $\Pr[Y > \gamma] \leq \delta$ at t and no alarm should be raised at this time instance; whenever

---

Algorithm Improved($c_1, \ldots, c_g, H$)
1. for $t = 1, \ldots, T$
2. let $X_i = X_{i,t}$ and $Y = Y_t = \sum_{i=1}^{g} X_i$;
3. each c_i computes $\mathrm{E}(X_i)$ and $\mathrm{Var}(X_i)$ locally, and sends them to H;
4. H sets $\mathrm{E}(Y) = \sum \mathrm{E}(X_i)$, $\mathrm{Var}(Y) = \sum \mathrm{Var}(X_i)$;
5. if ($\gamma > \mathrm{E}(Y)$ and $\frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y) + (\gamma - \mathrm{E}(Y))^2} \leq \delta$)
6. raise no alarm; **continue** to next time instance;
7. if ($\gamma < \mathrm{E}(Y)$ and $1 - \frac{\mathrm{Var}(Y)}{\mathrm{Var}(Y) + (\mathrm{E}(Y) - \gamma)^2} \geq \delta$)
8. raise an alarm; **continue** to next time instance;
9. H sets β_1 and β_2 according to (2.12);
10. H broadcasts β_1, β_2 to all clients, and asks them to compute and send back $M_i(\beta_1)'s$ and $M_i(\beta_2)'s$;
11. H sets $M(\beta_1) = \prod_i M_i(\beta_1)$, $M(\beta_2) = \prod_i M_i(\beta_2)$;
12. if ($e^{-\beta_1 \gamma} M(\beta_1) \leq \delta$)
13. raise no alarm; **continue** to next time instance;
14. if ($1 - e^{-\beta_2 \gamma} M(\beta_2) \geq \delta$)
15. raise an alarm; **continue** to next time instance;
16. H asks for X_i's, applies EXACTD or EXACTC;

Figure 2.2: The *Improved* method.

the second inequality in (2.13) holds at t, H knows for sure that $\Pr[Y > \gamma] > \delta$ at t and an alarm should be raised. Monitoring the first inequality in (2.13) is the same as monitoring if

$$\sum_{i=1}^{g} \ln M_i(\beta_1) \leq \ln \delta + \beta_1 \gamma. \tag{2.14}$$

We can treat $(\ln \delta + \beta_1 \gamma)$ as the global constraint, and at time t, let $V_i = \ln M_i(\beta_1)$ be the local deterministic score at client c_i; this becomes the exactly same formulation for the DTM problem. We now apply the adaptive thresholds algorithm for constraint monitoring from [48] to monitor (2.14). We denote this monitoring instance as J_1. At any time t, if J_1 raises no alarm, H knows that no alarm should be raised at t, since by implication (2.14) holds, and hence $\Pr[Y > \gamma] \leq \delta$.

Monitoring the 2nd inequality in (2.13) is to monitor if

$$\sum_{i=1}^{g} \ln M_i(\beta_2) \leq \ln(1 - \delta) + \beta_2 \gamma. \tag{2.15}$$

By treating $(\ln(1 - \delta) + \beta_2 \gamma)$ as the global constraint, at time t let $W_i = \ln M_i(\beta_2)$ be the local deterministic score at client c_i; then we again apply [48] to monitor (2.15). Denote this monitoring instance as J_2. Constrasting J_1 to J_2, when J_2 does *not* report an alarm at t, it means that (2.15) holds, which implies that $\Pr[Y > \gamma] > \delta$, so H needs to raise an alarm.

One choice is to let H run both J_1 and J_2. However, when $\Pr[Y > \gamma]$ deviates from δ considerably, one of them will almost always raise alarms, which results in a global poll and adjusting the local filters [48]. So the total communication cost will actually be higher than running just one. A critical challenge is deciding which instance to run. A simple and effective method is to make this decision periodically using recent observations of $\Pr[Y > \gamma]$ and δ.

Suppose we set the period to k, and the current time instance is t. For any $i \in [t - k, t)$, let $e_i = 1$ if $\Pr[Y_i > \gamma] > \delta$ and 0 otherwise; and $e = \sum_{i=t-k}^{t-1} e_i$. If $e \geq k/2$, then in a majority of recent instances $\Pr[Y_i > \gamma] > \delta$, hence (2.15) is more likely to hold and J_2 is most likely not going to raise alarms and is more efficient to run. If $e < k/2$, in a majority of recent instances $\Pr[Y_i > \gamma] < \delta$, (2.14) is more likely to hold and J_1 is most likely not going to raise alarms and is more efficient to run.

Another question is how to set the β_1 and β_2 values in (2.14) and (2.15). Since they are derived directly from (2.13), which are originated from (2.8) and (2.9), the same way of setting them as shown in (2.12) will likely result in tight bounds, thus less violations to (2.14) and (2.15), making J_1 and J_2 efficient to run, respectively. However, this does require H to ask for $\mathrm{E}(X_i)$'s and $\mathrm{Var}(X_i)$'s in every time instance, defeating the purpose of using the adaptive thresholds algorithm to reduce the number of messages. To remedy this, we let H reset the optimal β_1 and β_2 values for the two adaptive thresholds instances periodically in every k time instances, for a system parameter k.

The complete algorithm, *Iadaptive*, is shown in Figure 2.3.

2.5 Sampling Methods to Estimate the Threshold

In either of the previous methods, when the algorithm fails to definitively indicate whether an alarm should be raised or not, then likely $\Pr[Y > \gamma]$ is close to δ. If H

Algorithm Iadaptive($c_1, \ldots, c_g, H, k$)
1. initialize (without starting) two adaptive thresholds
   instances J_1, J_2 [48]: J_1 monitors $\sum_i V_i \leq \ln \delta + \beta_1 \gamma$,
   and J_2 monitors if $\sum_i W_i \leq \ln(1-\delta) + \beta_2 \gamma$;
2. H sets β_1 to a small positive value, $e = 0$, starts J_1;
3. for $t = 1, \ldots, T$
4. let $X_i = X_{i,t}$, $Y = Y_t = \sum X_i$;
5. c_i computes $V_i = \ln M_i(\beta_1)$, or $W_i = \ln M_i(\beta_2)$;
6. if (J_1 is running and raises no alarm)
7. H raises no alarm; **continue** to line 11;
8. if (J_2 is running and raises no alarm)
9. H raises an alarm; $e = e + 1$; **continue** line 11;
10. H asks for X_i's, applies EXACTD or EXACTC,
    sets $e = e + 1$ if an alarm is raised;
11. if ($t \mod k == 0$)
12. stop the currently running instance J_x;
13. each c_i sends $\mathrm{E}(X_i)$ and $\mathrm{Var}(X_i)$ to H;
14. reset β_1 in J_1 and β_2 in J_2 according to (2.12);
15. if ($e \geq k/2$) set $x = 2$ else set $x = 1$;
16. H sets $e = 0$, starts J_x, broadcasts setup
    information of J_x, and new β_1 and β_2 values;

Figure 2.3: The *Iadaptive* method.

needs to be sure that the (γ, δ) threshold is crossed, all of X_i have to be retrieved, and the exact algorithms in Section 2.3.1 are applied. But in a lot of situations, this is expensive and impractical, due to both the communication and computation costs involved. Since uncertainties naturally exist in probabilistic data, it is very likely that users are willing to approximate the conditions under which the server raises the alarm, if approximation guarantees can be provided.

2.5.1 The Random Sampling Approach

A natural choice and standard approximation is to leverage random sampling. We first introduce the RS algorithm in Figure 2.4.

Lemma 1. *The* RS *estimator satisfies* $\mathrm{E}(\widehat{p}(\gamma)) = \Pr[Y > \gamma]$, *and* $\Pr[|\widehat{p}(\gamma) - \Pr[Y > \gamma]| < \varepsilon] > \frac{3}{4}$.

Proof. Let $\varepsilon' = \varepsilon/2$, then κ in line 1 is $1/\varepsilon'^2$. Clearly, by lines 2-5, S is a random sample of Y with size $1/\varepsilon'^2$. Suppose Y's distribution is represented by a *multi-set* P of elements $P = \{y_1, \ldots, y_N\}$ for some imaginary, sufficiently large value $N \in \mathbb{Z}^+$. Let $r(\gamma)$ be the number of elements in P that is larger than γ, then $\Pr[Y > \gamma] = r(\gamma)/N$.

Let $p = 1/(\varepsilon'^2 N)$; we then define N i.i.d. random variables $Z_1, \ldots, Z_N$, such that $\Pr[Z_i = 1] = p$ and $\Pr[Z_i = 0] = 1 - p$. We associate Z_i with $y_i \in P$. Then, S can be viewed as being created by the following process: for each $i \in [1, N]$, insert y_i into S if $Z_i = 1$. For any γ, $s(\gamma)$ in line 6 is a random variable determined by the number of elements in P larger than γ (each sampled with probability p) in S. There are precisely $r(\gamma)$ such elements in P, and we denote them as $\{y_{\ell_1}, \ldots, y_{\ell_{r(\gamma)}}\}$, where $y_{\ell_i} \in P$. This means

---

Algorithm RS $(c_1, \ldots, c_g, t, H, \varepsilon)$
1. let $X_i = X_{i,t}$, $Y = Y_t = \sum_{i=1}^{g} X_i$, $S = \emptyset$, $\kappa = 4/\varepsilon^2$;
2. for $i = 1, \ldots, g$
3. send *random sample* $S_i = \{x_{i,1}, \ldots, x_{i,\kappa}\}$ of X_i to H;
4. For any $j \in [1, \kappa]$, H inserts $\sum_{i=1}^{g} x_{i,j}$ into S;
5. let $s(\gamma)$ be the number of elements in S greater than γ;
6. return $\widehat{p}(\gamma) = s(\gamma) \cdot \frac{\varepsilon^2}{4}$;

Figure 2.4: The RS estimator

that: $s(\gamma) = \sum_{i=1}^{r(\gamma)} Z_{\ell_i}$. Since each Z_i is a Bernoulli trial, $s(\gamma)$ is a Binomial distribution $B(r(\gamma), p)$. Immediately, $\mathrm{E}(s(\gamma)) = p \cdot r(\gamma)$. Hence, $\mathrm{E}(\widehat{p}(\gamma)) = \mathrm{E}(\varepsilon'^2 N \frac{s(\gamma)}{N}) = \frac{1}{p}\frac{p \cdot r(\gamma)}{N} = \Pr[Y > \gamma]$, and

$$\begin{aligned} \mathrm{Var}(\frac{s(\gamma)}{p}) &= \frac{1}{p^2}\mathrm{Var}(s(\gamma)) = \frac{1}{p^2} r(\gamma) p(1-p) \\ &< \frac{r(\gamma)}{p} = r(\gamma)\varepsilon'^2 N \leq (\varepsilon' N)^2. \end{aligned}$$

Also, $\mathrm{E}(s(\gamma)/p) = r(\gamma)$. By Chebyshev's inequality: $\Pr[\left|\frac{s(\gamma)}{p} - r(\gamma)\right| \geq 2\varepsilon' N] \leq \frac{1}{4}$, which implies that: $\Pr[\frac{1}{N}\left|\frac{s(\gamma)}{p} - r(\gamma)\right| \geq 2\varepsilon'] \leq \frac{1}{4}$. Given $\varepsilon = 2\varepsilon'$ and $p = 1/(\varepsilon'^2 N)$, $\frac{s(\gamma)}{pN} = \frac{s(\gamma)\varepsilon^2}{4}$, we have $\Pr[\left|\frac{s(\gamma)\varepsilon^2}{4} - \Pr[Y > \gamma]\right| \geq \varepsilon] \leq \frac{1}{4}$. Immediately, $\Pr[|\widehat{p}(\gamma) - \Pr[Y > \gamma]| < \varepsilon] > \frac{3}{4}$. □

We can boost up $\Pr[|\widehat{p}(\gamma) - \Pr[Y > \gamma]| < \varepsilon]$ to be arbitrarily close to 1 by the MRS (*m*edian RS) Algorithm in Figure 2.5.

Theorem 1. MRS *returns* $\widehat{p}_j(\gamma)$ *s.t.* $\Pr[|\widehat{p}_j(\gamma) - \Pr[Y > \gamma]| < \varepsilon] > 1 - \phi$*, for any* $\varepsilon, \phi \in (0, 1)$*; it uses* $32\frac{g}{\varepsilon^2}\ln\frac{1}{\phi}$ *bytes.*

Proof. By Lemma 1, each I_i outputs 1 with probability at least $\frac{3}{4}$ in line 3 in Figure 2.5. Let $h = 8\ln\frac{1}{\phi}$; by the common form of the Chernoff Bound [58], $\Pr[\sum_{i=1}^{h} I_i < \frac{h}{2}] < e^{-2h(\frac{3}{4}-\frac{1}{2})^2} = \phi$. $\Pr[\sum_{i=1}^{h} I_i < \frac{h}{2}]$ is exactly the probability of less than half of I_i's being 0. Since I_j is the median in I (line 4), there is at least $(1-\phi)$ probability that $I_j = 1$. By line 3, in this case, we must have $|\widehat{p}_j(\gamma) - \Pr[Y > \gamma]| < \varepsilon$. The communication in bytes is straightforward. □

Algorithm MRS $(c_1, \ldots, c_g, t, H, \varepsilon, \phi)$
1. run $8\ln\frac{1}{\phi}$ independent instances RS $(c_1, \ldots, c_g, t, H, \varepsilon)$;
2. let $\widehat{p}_i(\gamma)$ be the ith RS's output for $i \in [1, 8\ln\frac{1}{\phi}]$;
3. set I_i be 1 if $|\widehat{p}_i(\gamma) - \Pr[Y > \gamma]| < \varepsilon$, and 0 otherwise;
4. let I_j be the median of $I = \{I_1, \ldots, I_{8\ln\frac{1}{\phi}}\}$;
5. return $\widehat{p}_j(\gamma)$;

Figure 2.5: The MRS estimator

Lastly, if $\widehat{p}(\gamma)$ returned by MRS is greater than δ, H raises an alarm at t; otherwise, no alarm is raised. It approximates $\Pr[Y > \gamma]$ within ε with at least $(1-\phi)$ probability, using $O(g/\varepsilon^2 \ln(1/\phi))$ bytes, for any $\varepsilon, \phi \in (0,1)$.

2.5.2 Random Distributed ε-Sample

Instead of using the standard random sampling approach as shown in Section 2.5.1, we can leverage on a more powerful machinery in our analysis to derive a new algorithm with the same guarantee w.r.t. a fixed pair of thresholds (γ, δ), but it is simpler to implement and works better in practice. Later, in Section 2.6, we also show that it can handle multiple pairs of thresholds simultaneously without incurring additional costs.

We can approximate the probabilities of raising an alarm by a Monte Carlo approach where H asks each c_i for a sample x_i from X_i. He then computes a value $y = \sum_{i=1}^{g} x_i$; this is a sample estimate from the distribution over Y, so $\Pr[Y > \gamma] = \Pr[y > \gamma]$. Repeating this to amplify success is the *r*andom *d*istributed ε-*s*ample (RDεS) algorithm in Figure 2.6.

Theorem 2. RDεS *gives* $\mathrm{E}(v/\kappa) = \Pr[Y > \gamma]$ *and* $\Pr[|v/\kappa - \Pr[Y > \gamma]| \leq \varepsilon] \geq 1 - \phi$, *using* $O(\frac{g}{\varepsilon^2} \ln \frac{1}{\phi})$ *bytes.*

Proof. First, it is clear that in line 7 for any $j \in [1, \kappa]$, $y_j = \sum_{i=1}^{g} x_{i,j}$ is a random sample drawn from the distribution of Y. Hence, $\mathrm{E}(v) = \kappa \cdot \Pr[Y > \gamma]$.

We next leverage on the concept of *VC-dimension* [82]. Let P be a set of points, or

---

Algorithm RDεS $(c_1, \ldots, c_g, H, t, \varepsilon, \phi)$
1. $X_i = X_{i,t}$, $Y = \sum_{i=1}^{g} X_i$, $S_i = \emptyset$, $v = 0$, $\kappa = \frac{1}{\varepsilon^2} \ln \frac{1}{\phi}$;
2. for $i = 1, \ldots, g$
3. for $j = 1, \ldots, \kappa$
4. c_i selects some value $x_{i,j}$ from X_i, into S_i, at random according to its underlying distribution;
5. c_i sends S_i to H;
6. for $j = 1, \ldots, \kappa$
7. if $(y_j = \sum_{i=1}^{g} x_{i,j} > \gamma)$ $v = v + 1$;
8. if $(v/\kappa > \delta)$ H raises an alarm;
9. else H raises no alarm;

Figure 2.6: The RDεS method.

more generally a distribution. Let $\mathcal{I}$ be a family of subsets of P. Let P have domain $\mathbb{R}$ and let $\mathcal{I}$ consist of ranges of the form of one-sided intervals (x, ∞) for any $x \in \mathbb{R}$. The pair $(P, \mathcal{I})$ is a *range space* and we say a subset $X \subset P$ *shatters* a range space $(P, \mathcal{I})$ if every subset $X_s \subseteq X$ can be defined as $I \cap X$ for some $I \in \mathcal{I}$. The size of the largest subset X that shatters $(P, \mathcal{I})$ is the VC-dimension of $(P, \mathcal{I})$. For one-sided intervals $\mathcal{I}$, the VC-dimension for a range space $(P, \mathcal{I})$ using any set P is $\nu = 1$.

An ε-*sample* for a range space $(P, \mathcal{I})$ is a subset $Q \subset P$ that approximates the density of P such that:

$$\max_{I \in \mathcal{I}} \left| \frac{|I \cap P|}{|P|} - \frac{|I \cap Q|}{|Q|} \right| \leq \varepsilon. \tag{2.16}$$

A classic result of Vapnik and Chervonenkis [82] shows that if $(P, \mathcal{I})$ has VC-dimension ν and if Q is a random sample from P of size $O((\nu/\varepsilon^2)\log(1/\phi))$, then Q is an ε-sample of $(P, \mathcal{I})$ with probability at least $1 - \phi$.

Every y_j in line 7 can be viewed as a random point in P, the distribution of values for Y. The ranges we estimate are one-sided intervals ((γ, ∞) for any $\gamma \in \mathbb{R}$ and they have VC-dimension $\nu = 1$). If we let $\kappa = O((1/\varepsilon^2)\ln(1/\phi))$, DTS gets exactly an ε-sample and guarantees that $|v/\kappa - \Pr[Y > \gamma]| \leq \varepsilon$ with probability at least $1 - \phi$. □

2.5.3 Deterministic Distributed ε-Sample

The sizes of samples in RDεS could be large, especially for small ε and ϕ values, which drive up the communication cost (measured in bytes). We introduce another sampling algorithm, the *d*eterministic *d*istributed ε-*s*ample (DDεS) method, to address this problem, which is shown in Figure 2.7.

Let $\tilde{X}_i$ represent S_i in the DDεS algorithm. Clearly, $\tilde{X}_i$ approximates X_i. Let $\tilde{Y} = \sum_{i=1}^{g} \tilde{X}_i$, i.e., for any $u \in (1, \ldots, \kappa)^g$ (as in lines 6-8), insert $\sum_{i=1}^{g} x_{i,u_i}$ into $\tilde{Y}$; by the construction of the DDεS, it is easy to see that:

$$\Pr[\tilde{Y} > \gamma] = v/\kappa^g. \tag{2.17}$$

To analyze its error, consider the distribution $Y_{\neq j} = \sum_{i=1, i\neq j}^{g} X_i$. Note that $Y = Y_{\neq j} + X_j$. We can claim the following about the random variable $\tilde{Y}_j = Y_{\neq j} + \tilde{X}_j$:

Lemma 2. *If $\tilde{X}_j$ is an ε-sample of $(X_j, \mathcal{I})$ then $|\Pr[\tilde{Y}_j > \gamma] - \Pr[Y > \gamma]| \leq \varepsilon$ with*

Algorithm DDεS $(c_1, \ldots, c_g, H, t, \varepsilon, \phi)$
1. $X_i = X_{i,t}$, $Y = \sum_{i=1}^{g} X_i$, $S_i = \emptyset$, $v = 0$;
2. $\varepsilon' = \varepsilon/g$, $\kappa = 1/\varepsilon'$;
3. for $i = 1, \ldots, g$
4. $\quad c_i$ selects κ evenly-spaced $x_{i,j}$'s from X_i into S_i, s.t. $S_i = \{x_{i,1}, \ldots, x_{i,\kappa}\}$, and $\int_{x=x_{i,j}}^{x_{i,j+1}} Pr[X_i = x]dx = \varepsilon'$;
5. $\quad c_i$ sends S_i to H;
6. let $(1, \ldots, \kappa)^g$ define a g-dimensional space where each dimension takes values $\{1, \ldots, \kappa\}$;
7. for each $u \in (1, \ldots, \kappa)^g$ $\quad$ //u is a vector of g elements
8. $\quad$ if $(\sum_{i=1}^{g} x_{i,u_i} > \gamma)$ $v = v + 1$;
9. if $(v/\kappa^g > \delta)$ H raises an alarm;
10. else H raises no alarm;

Figure 2.7: The DDεS method.

probability 1.

Proof. The distribution of the random variable $\tilde{Y}_j$ has two components $Y_{\neq j}$ and $\tilde{X}_j$. The first has no error, thus,

$$\Pr[\tilde{Y}_j > \gamma] = \frac{1}{|\tilde{X}_j|} \sum_{x \in \tilde{X}_j} \Pr[x + Y_{\neq j} > \gamma]$$

Each $x \in \tilde{X}_j$ shifts the distribution of the random variable $Y_{\neq j}$, so part of that distribution that is greater than γ for $x_i \in \tilde{X}_j$ will also be greater than γ for $x_{i+1} \in \tilde{X}_j$ (since $x_{i+1} > x_i$ by definition). Let $y_i = \gamma - x_i$ denote the location in the distribution for $Y_{\neq j}$ where x_i causes $y \in Y_{\neq j}$ to have $\tilde{Y}_j > \gamma$. Now for $y \in [y_i, y_{i+1}]$ has $y + x_l \leq \gamma$ if $l < i$ and $y + x_l > \gamma$ if $l \geq i$. So $y \in [y_i, y_{i+1}]$ only has error in $\Pr[y + x > \gamma]$ (where x is either drawn from X_j or $\tilde{X}_j$) for $x \in [x_i, x_{i+1}]$. Otherwise, for $x \in [x_l, x_{l+1}]$, for $l < i$ has $\Pr[y + x > \gamma] = 0$ and for $x \in [x_l, x_{l+1}]$, for $l > i$ has $\Pr[y + x > \gamma] = 1$. Since for any i $\int_{x=x_i}^{x_{i+1}} \Pr[X_j = x] \leq \varepsilon$ (because $\tilde{X}_j$ is an ε-sample of $(X_j, \mathcal{I})$), we observe that:

$$\int_{y=y_i}^{y_{i+1}} \Pr[Y_{\neq j} = y] \tfrac{1}{|\tilde{X}_j|} \textstyle\sum_{x \in \tilde{X}_j} \mid \Pr[y + x > \gamma] - \Pr[y + X_j > \gamma] \mid dy$$
$$\leq \varepsilon \int_{y=y_i}^{y_{i+1}} \Pr[Y_{\neq j} = y]dy.$$ Thus, we use that

$$\Pr[\tilde{Y}_j > \gamma] = \int_y \Pr[Y_{\neq j} = y] \frac{1}{|\tilde{X}_j|} \sum_{x \in \tilde{X}_j} \Pr[y + x > \gamma] dy$$

to conclude that

$$\left| \Pr[Y > \gamma] - \Pr[\tilde{Y}_j > \gamma] \right| \leq \sum_{i=0}^{|\tilde{X}_j|} \varepsilon \int_{y=y_i}^{y_{i+1}} \Pr[Y_{\neq j} = y] dy \leq \varepsilon.$$

□

This bounds the error on Y with $\tilde{Y}_j$ where a single X_j is replaced with $\tilde{X}_j$. We can now define $(\tilde{Y}_j)_l = \tilde{Y}_j - X_l + \tilde{X}_l = \sum_{i=1 \neq j,l}^{g} X_i + \tilde{X}_j + \tilde{X}_l$, and then apply Lemma 2 to show that if $\tilde{X}_l$ is an ε-sample of $(X_l, \mathcal{I})$ then

$$|\Pr[(\tilde{Y}_j)_l > \gamma] - \Pr[\tilde{Y}_j > \gamma]| \leq \varepsilon.$$

We can apply this lemma g times, always replacing one X_i with $\tilde{X}_i$ in the approximation to Y. Then the sum of error is at most εg. This implies the following theorem.

Theorem 3. *If for each c_i constructs $\tilde{X}_i$ as an (ε/g)-sample for $(X_i, \mathcal{I})$ then for any γ $|\Pr[\tilde{Y} > \gamma] - \Pr[Y > \gamma]| \leq \varepsilon$ with probability 1.*

Finally, by the definition of ε-samples on one-sided intervals (refer to (2.16) and the fact that in our case $\mathcal{I}$ consists of (γ, ∞)'s), it is easy to see that:

Lemma 3. *Using g/ε evenly spaced points, each S_i in* DDεS *gives $\tilde{X}_i$ that is an ε/g-sample of $(X_i, \mathcal{I})$.*

Combining with (2.17), we have:

Corollary 1. DDεS *gives $|\Pr[\tilde{Y} > \gamma] - \Pr[Y > \gamma]| \leq \varepsilon$ with probability 1 in g^2/ε bytes.*

2.5.4 A Randomized Improvement

We can improve the analysis slightly by randomizing the construction of the α-samples for each X_i. We choose $x_{i,1} \in \tilde{X}_i$ (the smallest point) to be at random so that $\Pr[x_{i,1} = x] = \frac{1}{\alpha} \Pr[X_i = x \mid x \leq x_\alpha]$ where x_α is defined so $\int_{x=-\infty}^{x_\alpha} \Pr[X_i = x] dx = \alpha$. Then each $x_{i,j}$ still satisfies that $\int_{x \in x_{i,j}}^{x_{i,j+1}} \Pr[X_i = x] dx = \alpha$. This keeps the points evenly spaced, but randomly shifts them.

Now we can improve Theorem 3 by modifying the result of Lemma 2. We can instead state that the error caused by $\tilde{X}_i$

$$H_i = (\Pr[\tilde{Y}_j > \gamma] - \Pr[Y > \gamma]) \in [-\alpha, \alpha].$$

Because the random shift of $\tilde{X}_i$ places each $x_{i,j} \in \tilde{X}_i$ with equal probability as each point it represents in X_i, then for $I \in \mathcal{I}$ we have that

$$\mathrm{E}\left[\frac{|I \cap \tilde{X}_i|}{|\tilde{X}_i|}\right] = \mathrm{E}\left[\frac{|I \cap X_i|}{|X_i|}\right]$$

and hence for any γ $\mathrm{E}[\Pr[\tilde{Y}_j > \gamma]] = \mathrm{E}[\Pr[Y > \gamma]]$. Thus, $\mathrm{E}[H_i] = 0$ and for all i $\Delta = \max\{H_i\} - \min\{H_i\} \leq 2\alpha$. Since the H_i are independent, we can apply a Chernoff-Hoeffding bound to the error on $\tilde{Y}$. So,

$$\begin{aligned}\Pr[|\Pr[\tilde{Y} > \gamma] - \Pr[Y > \gamma]| \geq \varepsilon] &= \Pr[|\sum_{i=1}^{g} H_i| \geq \varepsilon] \\ &\leq 2\exp(-2\varepsilon^2/(g\Delta^2)) \leq 2\exp(-\varepsilon^2/(2g\alpha^2)) \leq \phi,\end{aligned}$$

when $\alpha \leq \varepsilon/\sqrt{2g\ln(2/\phi)}$. This implies that:

Theorem 4. *If each $\tilde{X}_i$ is of size $(1/\varepsilon)\sqrt{2g\ln(2/\phi)}$ and is randomly shifted, for any γ*

$$\Pr[|\Pr[\tilde{Y} > \gamma] - \Pr[Y > \gamma]| < \varepsilon] > 1 - \phi.$$

This gives a better bound when the acceptable failure probability ϕ satisfies $2\ln(2/\phi) < g$. We can modify DDεS according to Theorem 4 to get the αDDεS method:

Corollary 2. αDDεS *guarantees* $\Pr[|\Pr[\tilde{Y} > \gamma] - \Pr[Y > \gamma]| < \varepsilon] > 1 - \phi$ *for any* $\varepsilon, \phi, \gamma$ *in* $(g/\varepsilon)\sqrt{2g\ln(2/\phi)}$ *bytes.*

2.5.5 Practical Improvements

Whenever a sample is required at any time t, for both RDεS and DDεS algorithms when the local sample size $|S_i|$ at t has exceeded the size required to represent the distribution X_i, client c_i simply forwards X_i to the server and the server can generate the sample for X_i himself. This is a simple optimization that will minimize the communication cost.

For the DDεS algorithm (in both its basic version and the random-shift version), a

drawback is that its computation cost might become expensive for larger sample size or a large number of clients. In particular, executing its lines 7-10 requires the calculation of κ^g sums. In practice, however, we have observed that the DDεS algorithm can still give accurate estimation if we test only a small, randomly selected subset of possible combinations of local samples, instead of testing all κ^g combinations, i.e., in line 7, we randomly select $m < \kappa^g$ such u's and in line 9 we test v/m instead.

2.6 Extension

2.6.1 Weighted Constraint

Suppose the user is interested in monitoring $Y = \sum_{i=1}^{g} a_i X_i$, for some weights $\{a_1, \ldots, a_g\}$, $\forall a_i \in \mathbb{R}^+$. All of our results can be easily extended to work for this case. The *Improved* and *Iadaptive* methods can be adapted based on the observations that: 1) $\mathrm{E}(Y) = \sum_{i=1}^{g} a_i \mathrm{E}(X_i)$ and $\mathrm{Var}(Y) = \sum_{i=1}^{g} a_i^2 \mathrm{Var}(X_i)$; 2) $M(\beta) = \prod_{i=1}^{g} M_i(a_i\beta)$. The RD$\varepsilon$S and DD$\varepsilon$S algorithms can also be easily adapted. For any sample j, instead of checking if $\sum_{i=1}^{g} x_{i,j} > \gamma$, they check if $\sum_{i=1}^{g} a_i x_{i,j} > \gamma$, in line 7 and 8 of Figures 2.6 and 2.7, respectively. The exact methods can also be extended easily. The discrete case is trivial, and the continuous case leverages on the observation that $\varphi(\beta) = \prod_{i=1}^{g} \varphi(a_i\beta)$.

2.6.2 Handling Multiple (γ, δ) Thresholds

The other nice aspect of RDεS and DDεS is that after the server has gathered the samples S_i's from all clients and he wants to check another threshold pair (γ', δ'), he already has sufficient information. H re-executes lines 6-9 of RDεS or lines 6-10 of DDεS, with the new threshold pair (γ', δ'). The estimation of $\Pr[Y > \gamma']$ is again within ε of δ' with at least probability $1 - \phi$ and 1 for RDεS and DDεS, respectively, i.e., the same error ε and the failure probability ϕ (or 0) cover all possible pairs (γ, δ) simultaneously in RDεS (or DDεS). This is especially useful if there was a continuous set of threshold pairs $\Gamma \times \Delta$ such that any violation of $(\gamma, \delta) \in \Gamma \times \Delta$ should raise the alarm. Then RDεS and DDεS are sufficient to check all of them, and are correct within ε with probability at least $(1 - \phi)$ and 1, respectively, without additional costs.

This also means that RDεS delivers stronger guarantee than the basic random sampling method in Section 2.5.1. For the basic random sampling method approach, a second pair of thresholds (γ', δ') is a separate, but dependent problem. We can also estimate $\Pr[Y > \gamma'] >$

δ' with ε-error with failure probability ϕ using the same sample as we used for estimating $\Pr[Y > \gamma] > \delta$. But now the probability that either of the thresholds has more than ε error is greater than ϕ. Using union bound, we need a sample size of about $O(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon\phi})$ from each client to monitor $\frac{1}{\varepsilon}$ pairs of thresholds simultaneously, which is more than the sample size $O(\frac{1}{\varepsilon^2}\log\frac{1}{\phi})$ required by RDεS.

Small additional samples are also required for αDDεS to monitor multiple pairs of thresholds simultaneously.

2.7 Experiments

All algorithms were implemented in C++. We used the GMP library when necessary in calculating the moment generating function $M_i(\beta)$. We simulated the distributed clients and the server, and executed all experiments in a Linux machine with an Intel Xeon E5506 cpu at 2.13GHz and 6GB memory. Since the flat model is used, server-to-client communication is *broadcast* and client-to-server communication is *unicast*. The server-to-client broadcast counts as one message, regardless the number of clients. Every client-to-server transmission is one separate message, which may contain multiple values or a pdf. Score and probability values are both 4 bytes.

2.7.1 Datasets and Setup

We used real datasets from the SAMOS project [70]. Raw readings from the research vessel *Wecoma* were obtained which consist of approximately 11.8 million records observed during a 9-month interval in 2010, from March to November. Each record consists of the current time and date, and the wind direction (WD), wind speed (WS), sound speed (SS), and temperature (TEM) measurements which are observed roughly every second (sometimes in less than a second). The wind direction measures the directional degree of the wind. The wind speed and sound speed are measured in meters per second and the temperature is in degrees Celsius. We observed that some measurements were erroneous or missing, e.g., a temperature of 999 or -999 degrees Celsius. Currently in SAMOS, to reduce communication and processing costs, records are grouped every τ consecutive seconds (the *grouping interval*), then replaced by one record taking the average readings of these records on each measurement respectively, which obviously loses a lot of useful information.

Instead, we derive pdfs (one per measurement) for records in one grouping interval

and assign these pdfs to an attribute-level probabilistic tuple. There are different ways to derive a pdf for a measurement attribute, for example, [24, 25, 50], which is not the focus of this work. Without loss of generality and to ease the presentation, we simply generate a discrete pdf based on the frequencies of distinct values for a given measurement attribute: the probability of a distinct value is proportional to its frequency over the total number of records in the current grouping interval.

Four measurements lead to four datasets WD, WS, SS, and TEM, each with one probabilistic attribute. We were unable to obtain additional datasets of large raw readings from other research vessels, since in most cases they did not keep them after reporting the average readings per grouping interval. As a result, we simulate the effect of having multiple distributed vessels by assigning to each vessel tuples from a given dataset. Tuples are assigned in a round robin fashion to ensure and preserve the temporal locality of observed measurements.

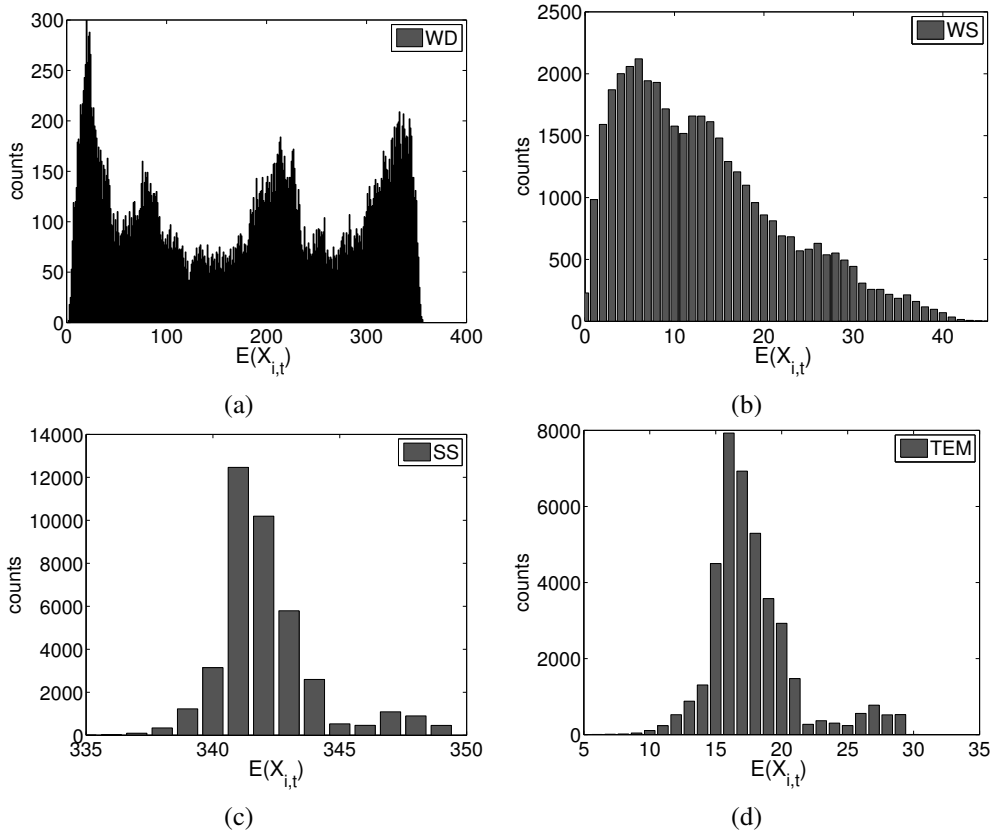


Figure 2.8: Distributions of $\mathrm{E}(X_{i,t})$ for WD, WS, SS, and TEM, where $i \in [1, g]$ and $t \in [1, T]$. (a) WD. (b) WS. (c) SS. (d) TEM.

The default values of key parameters are: $\tau = 300$, $g = 10$, $\delta = 0.7$, and γ is set to a value for a given dataset such that over all T instances, there should be approximately 30% alarms raised by an exact algorithm. The domains (in $\mathbb{R}$) of WD, WS, SS, and TEM are $[0, 359]$, $[0, 58.58]$, $[335.25, 355.9]$, and $[5.88, 41.3]$, respectively. These datasets also give us quite different distributions, allowing us to investigate different algorithms thoroughly. To illustrate this, we plot the distributions of $\mathrm{E}(X_{i,t})$ where $i = [1, g]$ and $t = [1, T]$ in the default setup in Figure 2.8. $\mathrm{E}(X_{i,t})$ also presents interesting (but quite different) temporal patterns and significant temporal changes in 4 datasets, which is also quite natural given that they precisely represent the large, real raw readings of different measurements at sea for a long period. Due to the space constraint, we omit these figures. That said, the default γ value is $230g$, $17g$, $343g$, and $19g$ for WD, WS, SS, and TEM. $X_{i,t}$ also has quite different sizes in 4 datasets. Under the default setup, the average size of $X_{i,t}$ is 41.15, 204.84, 20.5, and 20.98 for WD, WS, SS, and TEM, respectively (they also change when we vary τ, obviously). Under the default setup, $T = 3932$.

For each experiment, we vary one of the key parameters while keeping the others fixed at their default values. For any sampling method, the default *sample size per client* is $\kappa = 30$. In the *Iadaptive* method, $k = 0.3T$ by default. For communication costs and running time, since T may vary, we report *the average cost of one time instance* which is obtained by dividing the corresponding total cost by T. Note that, we calculate the total running time by counting the server's running time plus the maximum running time of one client at each time instance. This ensures that the average running time reflects the expected *response time* at each round (since clients are running in parallel at distributed sites).

When most $X_{i,t}$ have large variances, sampling methods have the worst approximations. In our datasets, $\mathrm{Var}(X_{i,t})$ in WD are consistently large (much larger than other datasets) which least favors our methods. WD also has a medium average distribution size and a wide range of values (which makes it the most interesting for a monitoring problem). Thus, we use WD as the default dataset. For our problem, the naive solution is to run EXACTD every time instance, which is clearly much worse than the two baseline methods, *Madaptive* and *Markov*. Between the two, *Madaptive* is always better. Hence, we only show the results from *Madaptive* as the competing baseline.

2.7.2 Effect of γ

Figure 2.9 shows the communication costs of *Madaptive*, *Improved*, and *Iadaptive* when we vary γ from 1500 to 3100. Both the number of messages and bytes reduce for all algorithms while γ increases, since probabilistic tail bounds become tighter for larger γ values. Nevertheless, Figure 2.9(a) indicates that *Iadaptive* communicates the least number of messages, and Figure 2.9(b) shows that *Improved* sends the least number of bytes. *Improved* employs the most sophisticated combination of various lower and upper bounds (on both sides of $\mathrm{E}(Y)$), thus it has the largest number of "certain" instances where retrieving $X_{i,t}$'s can be avoided, which explains its best communication cost in bytes. Furthermore, it maintains low bytes for all γ values (a wide range we have tested), meaning that its pruning is effective on both sides of $\mathrm{E}(Y)$. However, *Improved* does require at least one, to a few, message(s) per client at every time instance, as shown in Figure 2.9(a). When reducing the number of messages is the top priority, *Iadaptive* remedies this problem. Figure 2.9(a) shows in most cases, it uses only half to one-third the number of messages compared to *Madaptive* and *Improved*. In fact, it sends less than one message per client per time instances in most cases.

Figure 2.10(a) shows the response time of these methods when γ varies. Clearly, all methods take less time as γ increases, since there are less instances where they need to call the EXACTD method (which is costly). *Improved* and *Iadaptive* are much more efficient than *Madaptive*. The dominant cost in *Madaptive* and *Improved* is the calls to EXACTD, while the dominant cost in *Iadaptive* is the calculation of the moment generating

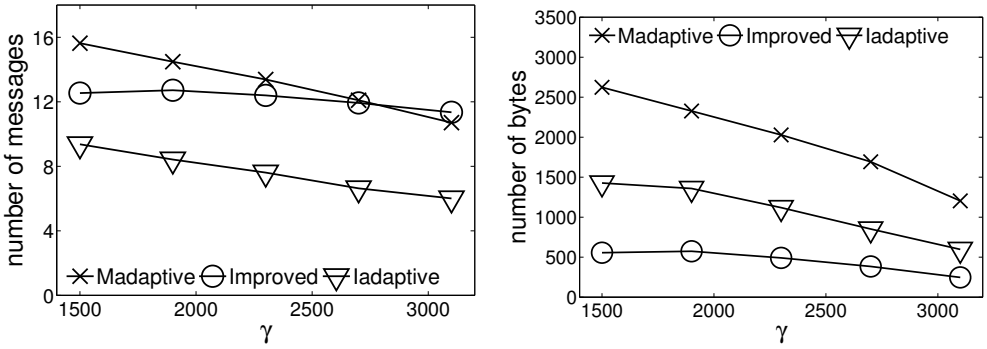


Figure 2.9: Communication: vary γ. (a) Messages. (b) Bytes.

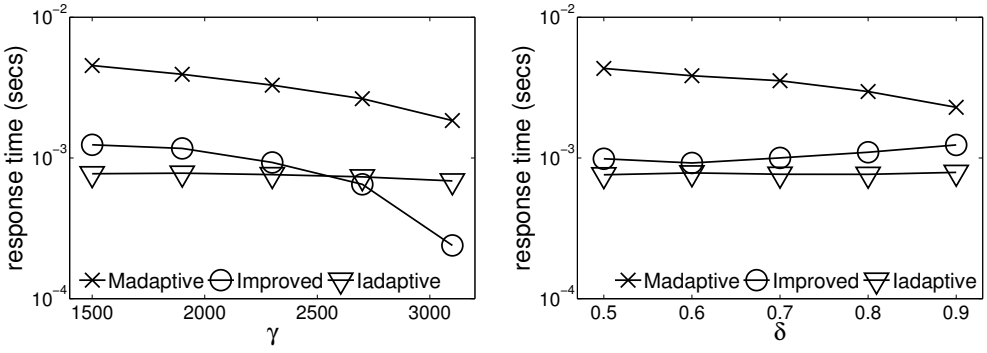


Figure 2.10: Response time: (a) vary γ. (b) vary δ.

function at the client. This explains why the response time of both *Madaptive* and *Improved* improves at a faster pace than that in *Iadaptive* when γ increases, since this mainly reduces the number of calls to EXACTD, but *Iadaptive* still needs to calculate moment generating functions. Nevertheless, *Iadaptive* is still more efficient than *Madaptive* in all cases. When $\gamma = 3100$, *Iadaptive* takes less than 0.001 second, and *Improved* takes close to 0.0003 second.

2.7.3 Effect of δ

When δ changes from 0.5 to 0.9 in Figure 2.11, *Madaptive* benefits the most where both its messages and bytes are decreasing, since its global constraint is linearly dependent on δ, leading to a linearly increasing global constraint. Nevertheless, *Iadaptive* still uses much fewer messages and bytes than *Madaptive*, and *Improved* uses the least number of bytes, in

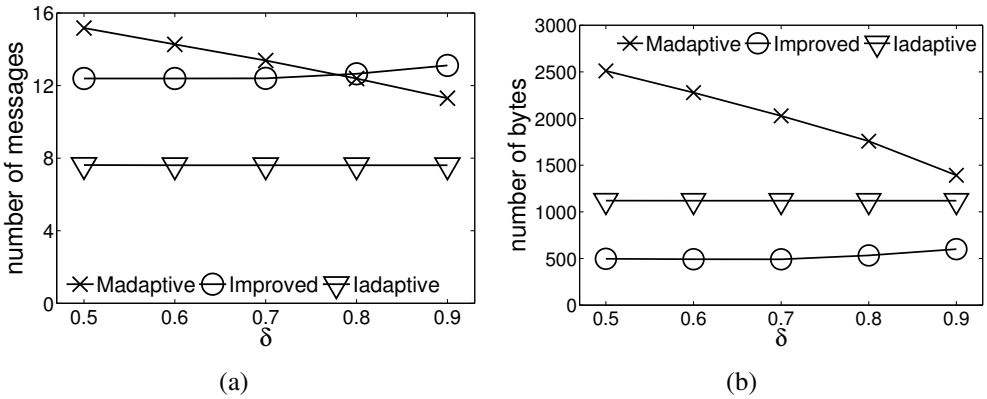


Figure 2.11: Communication: vary δ. (a) Messages. (b) Bytes.

all cases. In terms of the response time, Figure 2.10(b) shows that their trends are similar to what we have observed in Figure 2.10(a): *Improved* and *Iadaptive* are more efficient than *Madaptive*.

2.7.4 Effect of g

We next investigate the impact of the number of clients; Figure 2.12 shows the results on communication. Not surprisingly, we see a linear correlation between the number of messages and g in Figure 2.12(a) where *Iadaptive* consistently performs the best. Figure 2.12(b) shows that all methods send more bytes as g increases; nevertheless, both *Improved* and *Iadaptive* send many fewer bytes than *Madaptive*.

All methods take a longer to respond on average in Figure 2.13(a) for larger g values, due to the increasing cost in executing EXACTD. However, the cost of *Madaptive* increases

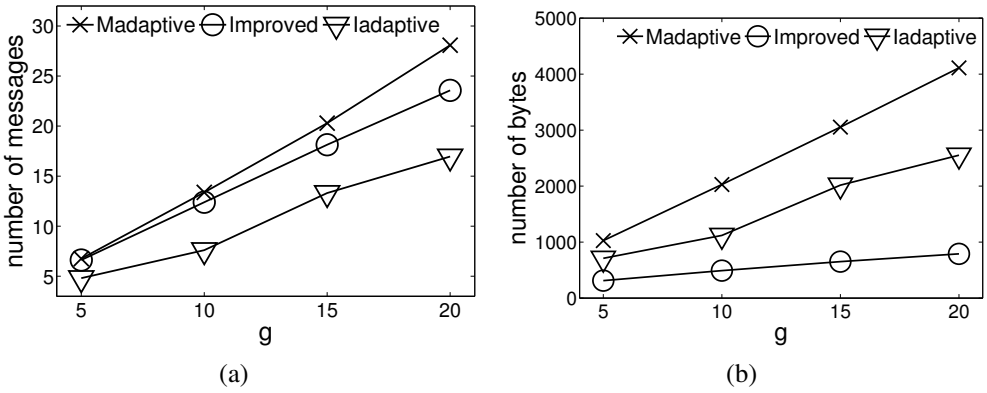


Figure 2.12: Communication: vary g. (a) Messages. (b) Bytes.

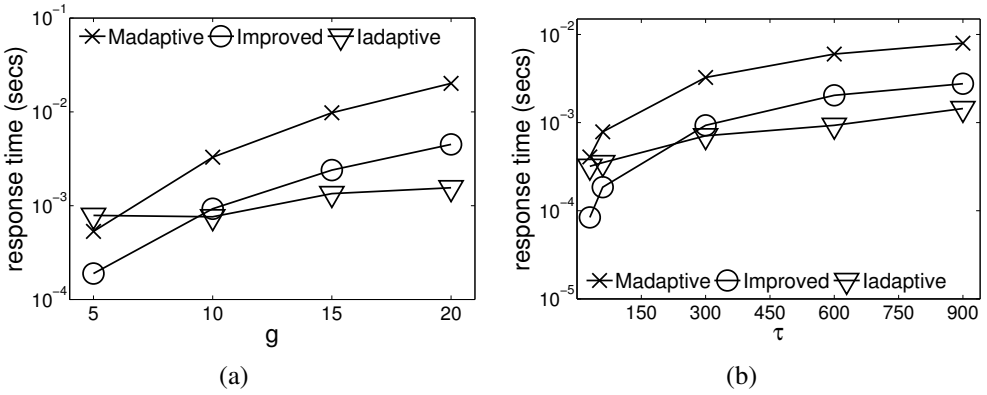


Figure 2.13: Response time: (a) vary g. (b) vary τ.

at a faster pace than other methods, since it makes many more calls to EXACTD. On the other hand, both *Improved* and *Iadaptive* are highly efficient, even though EXACTD becomes quite expensive for large g values, since they avoid calling EXACTD in most cases. Even when $g = 20$, both of them only take less than 0.005 seconds to respond.

2.7.5 Effect of τ

When τ changes, Figure 2.14 shows the communication of various methods. Figure 2.14(a) shows that *Iadaptive* reduces messages when τ increases, while the other two methods sends more messages. Larger τ values lead to larger pdfs, i.e., more values in $X_{i,t}$ but each taking smaller probability value, which make the bounds based on the moment generating functions tighter. But other bounds become looser, since $X_{i,t}$ becomes relatively more uniform for larger pdfs. Hence, *Iadaptive*, relying only the moment generating function bounds, is performing better for larger τ values, while others degrade slowly, in terms of number of messages. In terms of number of bytes, all methods send more bytes for larger τ values, which is easy to explain: whenever a call to EXACTD is necessary, $X_{i,t}$'s need to be communicated and they become larger for larger τ values. Nevertheless, both *Iadaptive* and *Improved* are still much more effective than *Madaptive*, e.g., even when $\tau = 900$ (15 minutes grouping interval), *Improved* only sends about 1000 bytes per time instance. Figure 2.13(b) shows that all methods take a longer time to respond, since EXACTD becomes more expensive due to the increase in the pdf size. *Improved* and *Iadaptive* are clearly faster than *Madaptive*. When $\tau = 900$, both of them still only take less than 0.005 second to respond.

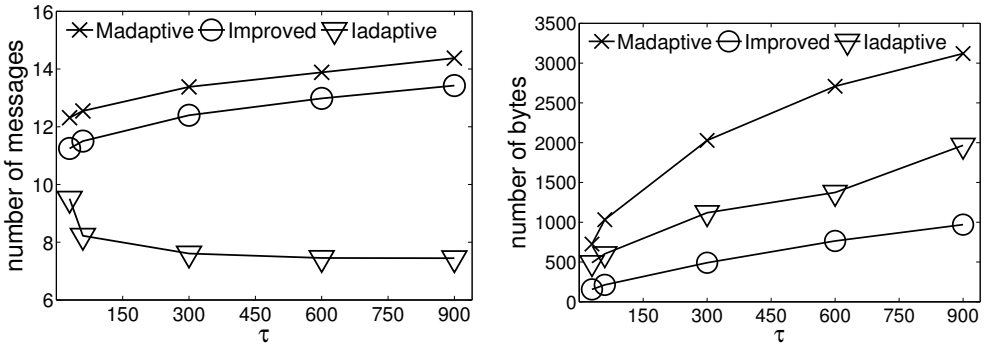


Figure 2.14: Communication: vary τ. (a) Messages. (b) Bytes.

2.7.6 Sampling Methods

The RDεS method offers similar (and even stronger, see Section 2.6.2) theoretical guarantee than the basic random sampling method in Section 2.5.1. Its performance in practice is also better. Thus, we focus on studying RDεS, DDεS, and its randomized improvement, denoted as αDDεS. Note that we have incorporated the practical improvements introduced in Section 2.5.5; $m = 2$ for both DDεS and αDDεS (which has achieved sufficient accuracy for both methods).

In this set of experiments, we compare sampling methods against the EXACTD method by running them over all T time instances. We use the *precision* and *recall* metrics to measure the approximation quality of sampling methods. Here, *precision* and *recall* are calculated w.r.t. the set of true alarms among the T instances, i.e., suppose there are a set A of 300 true alarms over $T = 1000$ time instances; an approximate method may raise a set B of 295 alarms out of the 1000 instances, with 5 false positives and 10 false negatives. Then, its precision is $290/295$ and its recall is $290/300$.

Figures 2.15(a) and 2.15(b) show that all sampling methods improve their precisions and recalls when the sample size per client κ increases. Theoretically, both αDDεS and DDεS should always have better precisions and recalls than RDεS given the same sample size. However, since we have incorporated the practical improvement to αDDεS and DDεS to cut down their computation cost, RDεS might perform better in some cases. Nevertheless, Figures 2.15(a) and 2.15(b) show that in practice, given the same sample size, αDDεS achieves the best precision while DDεS has the best recall; and αDDεS always outperforms RDεS. When $\kappa = 30$, they have achieved a precision and recall close to or higher than 0.98. The sample size required in practice to achieve good accuracy for all sampling methods is clearly much less than what our theoretical analysis has suggested. This is not surprising, since theoretical analysis caters for some worst cases that rarely exist in real datasets. In all remaining experiments, we use $\kappa = 30$ by default.

Figures 2.15(c) and 2.15(d) show that sampling methods result in clear savings in communication (bytes) and computation costs. They are especially useful in saving response time, which is 1-2 orders magnitude faster than EXACTD and the gap expects to be even larger for larger pdfs or more clients. Note that all sampling methods have the same communication cost given the same sample size (hence, we only show one line for all

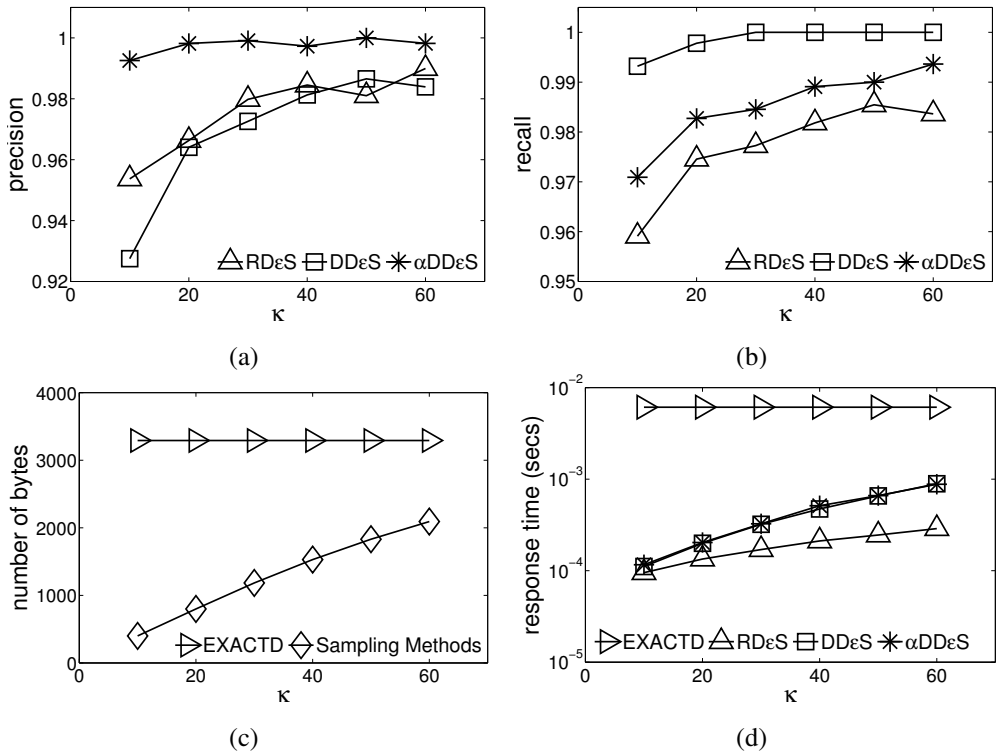


Figure 2.15: Performance of the sampling methods: vary κ (sample size per client). (a) Precision. (b) Recall. (c) Communication: bytes. (d) Response time.

of them in Figure 2.15(c)). Also, they result in the same number of messages as EXACTD.

We have also tested the sampling methods using all 4 datasets under the default setup, and the results are shown in Figure 2.16; the trends are clearly similar to what we have observed in Figure 2.15. Note that WS has quite large pdfs, thus, EXACTD becomes very expensive on this dataset in terms of both bytes communicated and running time, making sampling methods more valuable under these situations (several orders of magnitude more efficient than EXACTD).

2.7.7 Integrated Methods

Lastly, we integrate our sampling methods with *Madaptive*, *Improved*, and *Iadaptive* to derive the *MadaptiveS*, *ImprovedS*, and *IadaptiveS* methods, where in any time instance a call to EXACTD is replaced with a call to a sampling method. In particular, we use αDDεS as the sampling method since it achieves the best trade-off between efficiency and accuracy as shown in last set of experiments. We tested these methods, along with their exact versions, on all datasets using the default setup. The results are shown in Figure 2.17. The trends are

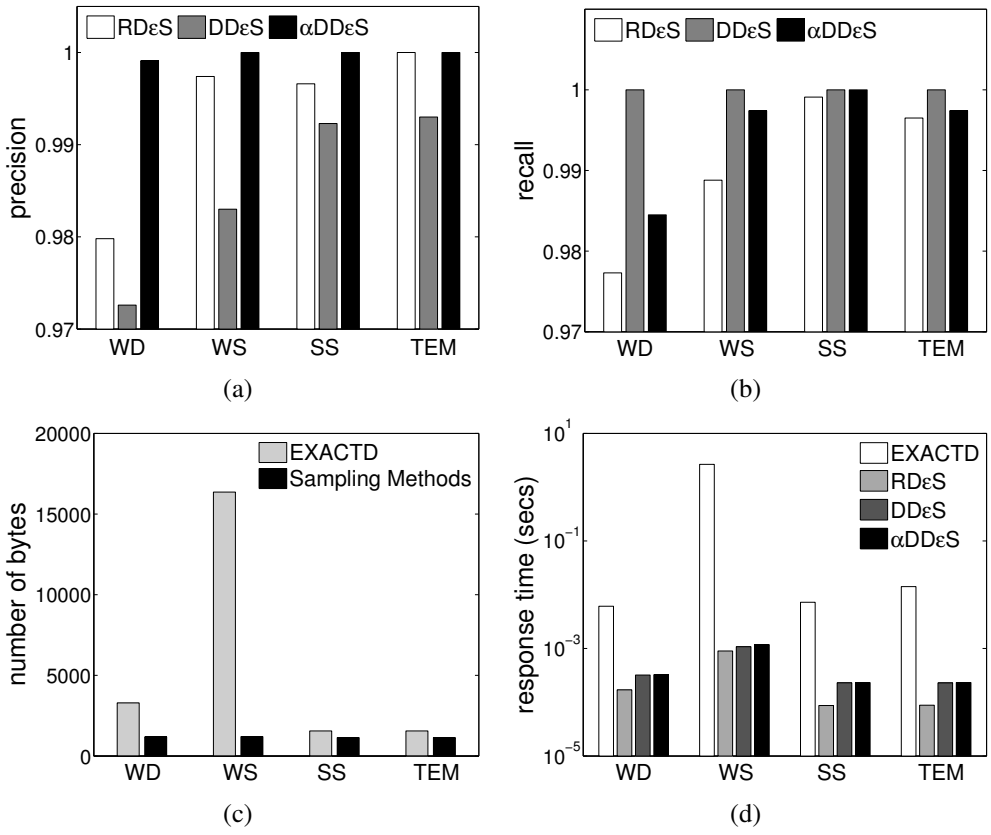


Figure 2.16: Performance of the sampling methods: vary datasets. (a) Precision. (b) Recall. (c) Communication: bytes. (d) Response time.

clear: 1) The approximate versions have outperformed the corresponding exact versions in both communication and response time consistently; 2) Our methods have outperformed the baseline methods, *Madaptive*, and *MadaptiveS* in all cases, by significant margins; 3) *Iadaptive* and *IadaptiveS* are the best exact and approximate methods in saving the number of messages, and *Improved* and *ImprovedS* are the best methods in saving the number of bytes. For example, *Iadaptive* and *IadaptiveS* use less than one message per client per time instance on all datasets; *Improved* and *ImprovedS* use less than 1000 and 100 bytes per time instance, respectively, on WS that has an average pdf size of 204.84; 4) *Iadaptive*, *IadaptiveS*, *Improved*, and *ImprovedS* are efficient to run. In particular, *IadaptiveS* and *ImprovedS* are extremely fast, e.g., Figure 2.17(c) shows that they take less than 10^{-3} and 10^{-4} seconds to respond, respectively, in all datasets. 5) αDDεS is highly effective. Figure 2.17(d) shows that *MadaptiveS*, *ImprovedS*, and *IadaptiveS* have almost perfect precisions and recalls on all datasets (more than 0.996 in all cases). Note that their precisions and

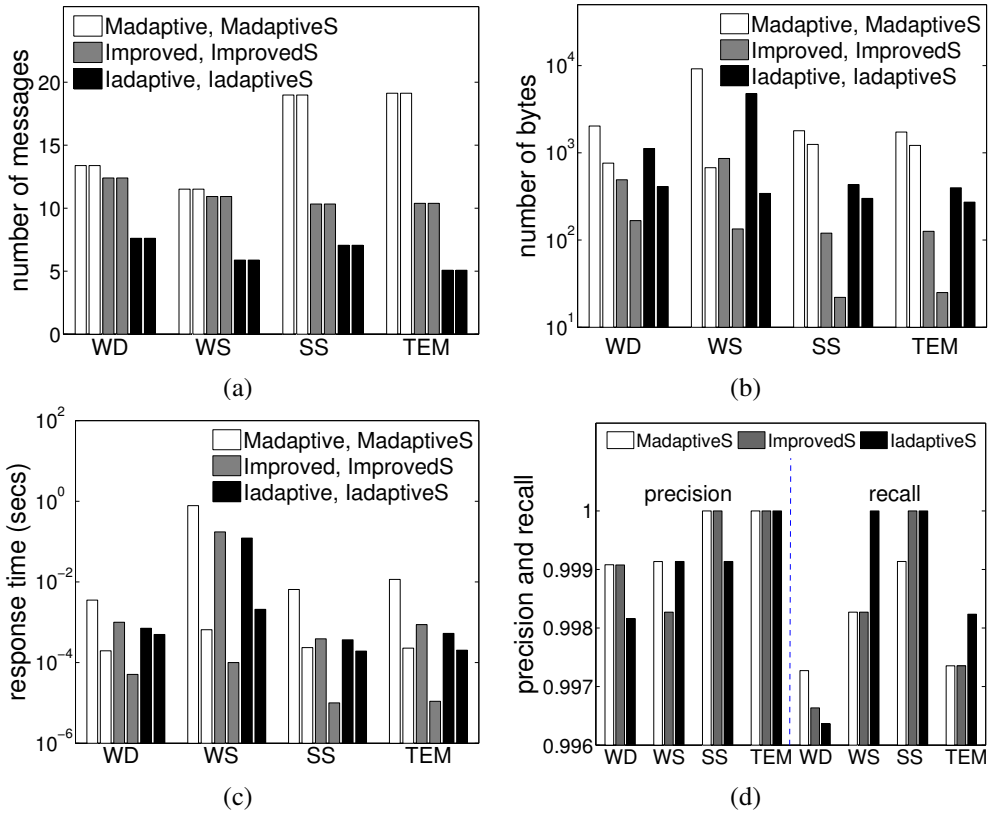


Figure 2.17: Performance of all methods: vary datasets. (a) Communication: messages. (b) Communication: bytes. (c) Response time. (d) Precision and recall.

recalls are clearly better than using sampling methods on every time instance; since many alarms will already be caught certainly by *Madaptive*, *Improved*, and *Iadaptive*, only a tiny fraction of undecided cases will be then decided by the sampling methods.

2.8 Related Work

To our knowledge, aggregate constraint monitoring on distributed data with uncertainty has not been explored before.

That said, ranking and frequent items queries were studied on distributed probabilistic data in [54, 83]. Monitoring centralized uncertain data for top-k and similarity queries were studied in [38, 49, 84]. On the other hand, due to their importance and numerous applications, constraint and function monitoring with thresholds on deterministic distributed data were examined extensively, e.g., [18, 39, 48, 52, 62, 74]. In our study, we have leveraged on the adaptive thresholds algorithm for the deterministic (sum) constraint monitoring from

[48]. This choice is independent from the design of our adaptive algorithms for the DPTM problem: any adaptive algorithms for the (sum) constraint monitoring in deterministic data can be used in our *Iadaptive* method.

Our study is also related to aggregation queries in probabilistic data, e.g., [45, 46, 60, 69, 76, 81]. However, monitoring both score and probability thresholds on aggregate constraints continuously over distributed probabilistic data is clearly different from these studies. Probabilistic threshold queries in uncertain data are also relevant [11, 22, 66, 68], as they are also concerned with the probability thresholds on the query results, but they mostly focus on one-shot query processing over centralized, offline probabilistic data.

Lastly, the basic sampling method MRS in Section 2.5.1 can be viewed as a standard extension of the random sampling technique [58, 82]. The RDεS and DDεS methods are related to VC-dimensions and ε-samples [82] as we already pointed out. The design principle behind the RDεS method, i.e., using a Monte Carlo approach, has also been used for general query processing in probabilistic data (e.g., [33, 44, 66] and more in [77]). The DDεS and αDDεS are based on several intriguing insights to the distinct properties of our problem.

2.9 Conclusion

We studied the threshold monitoring problem over distributed probabilistic data. We focused on continuously monitoring threshold constraint over the sum function of distributed probabilistic data and explore a number of novel methods that have effectively and efficiently reduced both the communication and computation costs. Extensive experiments demonstrate the excellent performance and significant savings achieved by our methods, compared to the baseline algorithms. Many interesting directions are open for future work. Examples include but are not limited to how to extend our study to the hierarchical model that is often used in a sensor network, how to continuously monitor a function value (e.g., max, min, median) of distributed data, and how to handle the case when data from different sites are correlated.

In the next chapter, we are going to study the distributed online tracking problem, which is exploring continuously functions tracking on a general-tree topological model.

CHAPTER 3

DISTRIBUTED ONLINE TRACKING

3.1 Introduction

The increasing popularity of smart mobile devices and the fast growth in the deployment of large measurement networks generate massive distributed data continuously. For example, such data include, but are not limited to, values collected from smart phones and tablets [8], measurements from large sensor-based measurement networks [26, 55, 86], application data from location-based services (LBS) [73], and network data from a large infrastructure network.

Tracking a user function over such distributed data *continuously* in an online fashion is a fundamental challenge. This is a critical task in many useful applications in practice. For example, it is a common task for users to continuously track the maximal (minimal) value of the temperature readings from a number of measurement stations. Similar examples can be easily found in location-based services and other distributed systems. This problem is also useful in the so-called publish/subscribe systems [9, 27], where a subscriber (tracker) may register a function (also known as a query) with a publisher (observer). Data continuously arrive at the publisher. The publisher needs to keep the subscriber informed about the value of her function f, when f is continuously applied over the *current data value*. When a subscriber's function of interest depends on data values from multiple publishers, it becomes a distributed tracking problem.

It is always desirable, sometimes even critical, to reduce the amount of communication in distributed systems and applications, for a number of reasons [3, 15–17, 26, 55, 56, 63, 86]. Many devices rely on on-board battery and incur high power consumption when they communicate, e.g., in sensors and smart phones. Hence, reducing the number of messages they need to send helps extend their battery time. Another reason is to save the network bandwidth. From the user's point of view, less communication often leads to economic

gains, e.g., most smart phones have a monthly budget for their data plan, or for nodes in remote areas in a large measurement network, communication via satellites come with a high price tag. From the network infrastructure's point of view (e.g., ISP such as Comcast), too much communications from any application could significantly congest their network and slow down the performance of the network (keep in mind that there could be many user applications running at the same time that share the available network bandwidth).

To achieve 100% accuracy for *continuous online tracking* of arbitrary functions, the solution is to ask all stations to always send their readings back to a centralized coordinator (the tracker), from which various functions can be easily computed and then tracked. This baseline approach, unfortunately, generates excessive communications: every new reading from any station must be forwarded to the tracker to ensure the correctness of the output values of the function being tracked.

But the good news is that, in many application scenarios, exact tracking is often unnecessary. Users are willing to trade-off accuracy with savings in communication. In some applications, approximation is often necessary not just for reducing communication, but also for policy constraints, e.g, due to privacy concerns in location-based services [4] and law requirements.

To formalize this accuracy and communication trade-off, we refer to a distributed site that continuously receives data from a data source as an *observer*, and the centralized site that wants to track a function (or multiple functions) computed over data from multiple, distributed data sources as the *tracker*. Without loss of generality, we assume that the tracker is tracking only one function, which is f. Clearly, f's output is a function of time, and is denoted as $f(t)$ for a time instance t. More precisely, it is a function of multiple data values at time instance t, one from each observer. Based on the above discussion, producing the exact values of $f(t)$ continuously for all time instances is expensive. Thus, the tracker's goal is to maintain an approximation $g(t)$, which is his best knowledge of $f(t)$ at any time instance t using a small amount of communication (accumulated so far). Focusing on functions that produce a one-dimensional output, we require that $g(t) \in [f(t)-\Delta, f(t)+\Delta]$ for any $t \in [0, t_{\text{now}}]$, for some user-defined error threshold Δ.

Under this set up, when $\Delta = 0$, $g(t)$ always equals $f(t)$ and the baseline exact solution is needed, which is communication-expensive. On the other hand, in the extreme case when

$\Delta = +\infty$, $g(t)$ can be a random value, and effectively there will be no communication needed at all. These two extremes illustrate the possible accuracy-communication trade-off enabled by this framework.

3.1.1 Key Challenge

It is important to note that our problem is a *continuous online* problem that requires a good approximation for *every time instance*. This is different from many distributed tracking problems in the literature that use the popular *distributed streaming model*, where the goal is to produce an approximation of certain functions/properties computed over the *union of data stream elements seen so far* for all observers, from the beginning of the time until now. It is also different from many existing work on monitoring a function over distributed streams, where the tracker only needs to decide if $f(t)$'s value has exceeded a given (constant) threshold or not at any time instance t.

When there is only one observer, our problem degenerates to a centralized, two-party setting (observer and tracker). This problem has only been recently studied in [87, 89] where they have studied both one-dimensional and multidimensional online tracking in this centralized setting. They have given an online algorithm with $O(\log \Delta)$ competitive ratio, and shown that this is optimal. In other words, any online algorithm for solving this problem must use at least a factor of $O(\log \Delta)$ more communication than the offline optimal algorithm. Note that, however, this problem is different from the classic problem of two-party computation [85] in communication complexity. For the latter problem, two parties Alice and Bob have a value x and y, respectively, and the goal is to compute some function $f(x, y)$ by communicating the minimum number of bits between them. Note that in online tracking, only Alice (the observer) sees the input; Bob (the tracker) just wants to keep track of it. Furthermore, in communication complexity both inputs x and y are given in advance, and the goal is to study the worst-case communication; in online tracking, the inputs arrive in an online fashion and it is easy to see that the worst-case (total) communication bound for online tracking is meaningless, since the function f could change drastically at each time step. For the same reasons, our problem, distributed online tracking, is also different from distributed multiparty computation.

3.1.2 Our Contributions

In this work, we extend the online tracking problem that was only recently studied in [87, 89] to the distributed setting with common aggregation functions (e.g., MAX), and investigate principled methods with formal theoretical guarantees on their performance (in terms of communication) when possible. We design novel methods that achieve good communication costs in practice, and formally show that they have good approximation ratios.

Our contributions are summarized below.

- We formalize the distributed online tracking problem in Section 3.2 and review the optimal online tracking method from [87, 89] in the centralized setting.
- We examine a special extension of the centralized setting with one observer but many relaying nodes, known as the *chain case*. We study the chain model in Section 3.3 and design a method with $O(\log \Delta)$ competitive ratio. We also show that our method has achieved the optimal *competitive ratio* in this setting.
- We investigate the "broom" model in Section 3.4 by leveraging our results from the chain model, where there are m distributed observers at the leaf-level and a single chain connecting them to the tracker. We design a novel method for MAX function and show that our method has very good approximation ratio among the class of online algorithms for the broom model.
- We extend our results to the general-tree model in Section 3.5, which is an extension of the broom model. We again show that our method has good approximation ratio among the class of online algorithms for the general-tree model.
- We discuss other functions and topologies in Section 3.6.
- We conduct extensive experiments to evaluate the effectiveness of our methods in practice in Section 4.6. We used several real datasets and the results have confirmed that our methods are indeed superior compared to other alternatives and baseline methods.

3.2 Problem Formulation and Background

Formally, there are m observers $\{s_1, \ldots, s_m\}$ at m distributed sites, and a tracker T. These m observers are connected to T using a network topology. We consider two common

topologies in this work, the *broom* topology and the *general-tree* topology, as shown in Figure 3.1. Observers always locate at the leaves, and the tracker always locates at the root of the tree. Both topologies are constructed based on a *chain topology*, as shown in Figure 3.2(a), and the centralized setting studied in [87, 89] is a special case of the chain topology, as shown in Figure 3.2(b). A relay node does not directly observe a function (or equivalently data values) that contribute to the computation of f, but it can receive messages from its child (or preceding) node(s), and send messages to its parent (or succeeding) node.

It is important to note that our general-tree topology has *already covered the case* in which an intermediate replay node u may be an observer at the same time, who also observes values (modeled by a function) that contribute to the computation of function f. This is because we can always conceptually add an observer node s directly below (and connected to) such an intermediate node u. Let s report the data values that are observed by u; we can then only view u as a relay node (while making no changes to all other connections to u that already exist). More details on this issue will be presented in Section

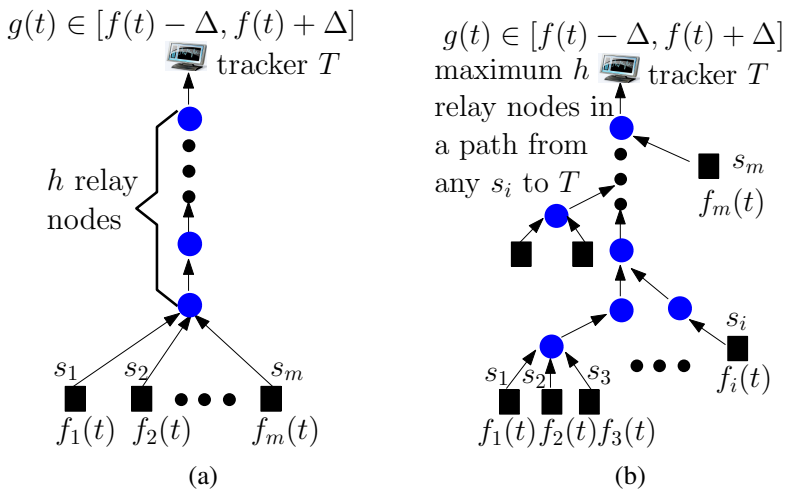


Figure 3.1: Track $f(t) = f(f_1(t), f_2(t), \cdots, f_m(t))$. (a) broom model. (b) general-tree.

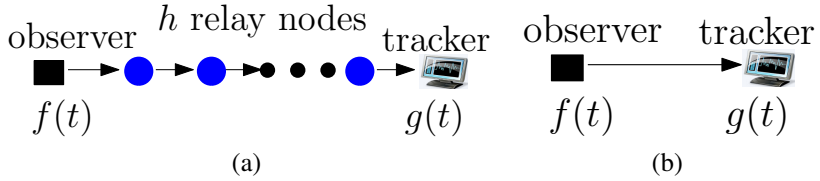


Figure 3.2: Special cases: $g(t) \in [f(t) - \Delta, f(t) + \Delta]$. (a) chain topology (b) centralized seting. [87, 89].

3.6.

That said, in practice, a relay node can model a router, a switch, a sensor node, a computer or computation node in a complex system (e.g., Internet, peer-to-peer network), a measurement station in a monitoring network, etc.

Each observer's data value changes (arbitrarily) over time, and can be described by a function. We dub the function at the ith observer f_i, and its value at time instance t $f_i(t)$. The tracker's objective is to continuously track a function f that is computed based on the values of functions from all observers at time instance t, i.e., its goal is to track $f(t) = f(f_1(t), f_2(t), \ldots, f_m(t))$ continuously over all time instances. Since tracking $f(t)$ exactly is expensive, an approximation $g(t)$ is allowed at the tracker T, subject to the constraint that $g(t) \in [f(t) - \Delta, f(t) + \Delta]$ for any time instance $t \in [0, t_{\text{now}}]$.

$\Delta \in \mathbb{Z}^+$ is a user-defined error threshold that defines the maximum allowed error in approximating $f(t)$ with $g(t)$. The goal is to find an online algorithm that satisfies this constraint while minimizing the communication cost.

Note that depending on the dimensionality for the outputs of $f(t)$, as well as $f_1(t)$, $f_2(t)$, ..., and $f_m(t)$, we need to track either a one-dimensional value or a multidimensional value that changes over time. This work focuses on the one-dimensional case. In other words, we assume that $f(t)$, and $f_1(t), \ldots, f_m(t)$ are all in a one-dimensional space.

3.2.1 Performance Metric of an Online Algorithm

There are different ways to formally analyze the performance of an online algorithm.

For an online problem P (e.g., caching), let $\mathcal{I}$ be the set of all possible valid input instances, and $\mathcal{A}$ be the set of all valid online algorithms for solving the problem P. Suppose the optimal *offline* algorithm for P is $\mathsf{offline}$. Given an input instance $I \in \mathcal{I}$, and an algorithm $A \in \mathcal{A}$ (or $\mathsf{offline}$), we denote the cost of running algorithm A on I as $\mathsf{cost}(A, I)$. In our setting, the cost is the total number of messages sent in a topology.

A widely used metric is the concept of *competitive ratio*. Formally, for an algorithm $A \in \mathcal{A}$, the competitive ratio of A [59], denoted as $\mathsf{cratio}(A)$, is defined as:

$$\mathsf{cratio}(A) = \max_{I \in \mathcal{I}} \frac{\mathsf{cost}(A, I)}{\mathsf{cost}(\mathsf{offline}, I)}.$$

Another popular metric is to analyze the performance of an algorithm A compared to

other algorithms in a class of online algorithms. Formally, we can define the *ratio* of A on an input instance I as follows:

$$\mathsf{ratio}(A, I) = \frac{\mathsf{cost}(A, I)}{\mathsf{cost}(A^*_I, I)},$$

where A^*_I is the online algorithm from the class $\mathcal{A}$ that has the lowest cost on input I, i.e., $A^*_I = \operatorname{argmin}_{A' \in \mathcal{A}} \mathsf{cost}(A', I)$.

Lastly, we can quantify an algorithm A's performance by considering its worst case *ratio*, i.e.,

$$\mathsf{ratio}(A) = \max_{I \in \mathcal{I}} \mathsf{ratio}(A, I).$$

Note that the definitions of $\mathsf{ratio}(A, I)$ and $\mathsf{ratio}(A)$ are inspired by the classic work that has motivated and defined the concept of "instance optimality" [29]. In fact, if $\mathsf{ratio}(A)$ is a constant, then indeed A is an *instance optimal online algorithm*.

Clearly, we always have, for any online problem P and its online algorithm A, $\mathsf{cratio}(A) \leq \mathsf{ratio}(A)$.

3.2.2 State-of-the-art Method

Prior work has studied the online tracking problem in the *centralized, two party setting* [87,89], as shown in Figure 3.2(b). They studied both one-dimensional tracking and multi-dimensional tracking, defined by the dimensionality of the output value for the function $f(t)$ at the observer. Since we focus on the one-dimension case, here we only review the one-dimension tracking method from [87, 89]. Finding a good online algorithm for this seemingly very simple setup turns out to be a very challenging problem.

Consider the simple case where the function takes integer values at each time step, i.e., $f : \mathbb{Z} \to \mathbb{Z}$, and the tracker requires an absolute error of at most Δ. The natural solution is to let the observer first communicate $f(t_0)$ to the tracker at the initial time instance t_0; then every time $f(t)$ has changed by more than Δ since the last communication, the observer updates the tracker with the current value of $f(t)$. However, this natural solution has an unbounded competitive ratio compared with the offline optimal method. Consider the case where $f(t)$ starts at $f(0) = 0$ and then oscillates between 0 and 2Δ. The above algorithm will communicate for an infinite number of times while the offline optimal solution only

needs to send one message: $g(0) = \Delta$.

This example demonstrates the hardness of the online tracking problem. For functions in the form of $f : \mathbb{Z} \to \mathbb{Z}$, Yi and Zhang proposed the method in Algorithm 1, and showed the following results.

Theorem 5. *(from [87, 89]) To track a function $f : \mathbb{Z} \to \mathbb{Z}$ within error Δ, any online algorithm has to send $\Omega(log\Delta \cdot OptHist)$ messages in the worst case, where $OptHist$ is the number of messages needed by the optimal offline algorithm. And,* OPTTRACK *is an $O(\log \Delta)$-competitive online algorithm to track any function $f : \mathbb{Z} \to \mathbb{Z}$ within Δ. Furthermore, if f takes values from the domain of reals (or any dense set), the competitive ratio of any online algorithm is unbounded.*

Theorem 5 establishes the optimality of the OPTTRACK method, since it shows that any online algorithms for centralized online tracking (between two nodes) has a competitive ratio that is at least $\log \Delta$, and OPTTRACK's competitive ratio $O(\log \Delta)$ has met this lower bound.

Note that the negative results on real domains and other dense domains do not rule out the application of OPTTRACK in practice on those cases. In practice, most functions (or data values for a sequence of inputs) have a fixed precision, e.g., any real number in a 64-bit machine can be described by an integer from an integer domain with size 2^{64}.

To the best of our knowledge, and as pointed out in [87, 89], no prior work has studied the distributed online tracking problem as we have formalized earlier in this section.

Algorithm 1: OPTTRACK (Δ) (from [87, 89])

```
let S = [f(t_now) − Δ, f(t_now) + Δ] ∩ ℤ
while S ≠ ∅ do
    let g(t_now) be the median of S;
    send g(t_now) to tracker T; set t_last = t_now;
    wait until |f(t) − g(t_last)| > Δ;
    S ← S ∩ [f(t) − Δ, f(t) + Δ]
```

3.3 The Chain Case

We first examine a special case that bridges centralized and distributed online tracking. Considering the tree topology in Figure 3.1, it is easy to observe that each observer is connected to the tracker via a single path with a number of relay nodes (if multiple paths exist, we simply consider the shortest path). Hence, online tracking in the chain topology as shown in Figure 3.2(a) is a basic building block for the general distributed online tracking problem. We refer to this problem as the *chain online tracking*.

The centralized online tracking as reviewed in Section 4.6.2 and shown in Figure 3.2(b) is a special case of chain online tracking, with 0 relay node.

3.3.1 Baseline Methods

For a chain topology with h relay nodes, a tempting solution is to distribute the error threshold Δ equally to all relay nodes and apply $(h+1)$ independent instances of the OPTTRACK algorithm. Suppose we have h relay nodes $\{n_1, \ldots, n_h\}$, an observer s, and a tracker T. Let $n_0 = s$ and $n_{h+1} = T$, for every pair of nodes $\{n_{i-1}, n_i\}$ for $i \in [1, h+1]$, we can view n_i as a tracker and its preceding node n_{i-1} as an observer, and require that n_i tracks n_{i-1}'s function be within an error threshold of $\frac{\Delta}{h+1}$.

Let $y_i(t)$ be the function at n_i for $i \in [1, h+1]$, then $y_i(t)$ is the output of running OPTTRACK with an error threshold $\frac{\Delta}{h+1}$, where n_{i-1} is the observer, $y_{i-1}(t)$ is the function to be tracked, and n_i is the tracker. Since $n_0 = s$ and $y_0(t) = f(t)$, we have two facts:

(1) $y_1(t) \in [f(t) - \frac{\Delta}{h+1}, f(t) + \frac{\Delta}{h+1}]$ for any time instance t.

(2) $y_i(t) \in [y_{i-1}(t) - \frac{\Delta}{h+1}, y_{i-1}(t) + \frac{\Delta}{h+1}]$ for any $i \in [2, h+1]$ and any time instance t.

Since the tracker T is simply node n_{h+1}, thus, $g(t) = y_{h+1}(t)$. Using the facts above, it is easy to verify that $g(t)$ will be always within $f(t) \pm \Delta$ as required. We denote this method as CHAINTRACKA (chain-track-average).

We can generalize CHAINTRACKA to derive other similar methods. Instead of distributing the error threshold uniformly along the chain, one can distribute a random error threshold Δ_i to node n_i for $i = 1, \ldots, h+1$, subject to the constraint that $\sum_{i=1}^{h+1} \Delta_i = \Delta$. We denote this method as CHAINTRACKR (chain-track-random). Using a similar argument, CHAINTRACKR also ensures that $g(t)$ at T is always within $f(t) \pm \Delta$.

Unfortunately, these seemingly natural solutions do not perform well, even though

they are intuitive extensions of the optimal online tracking method between two nodes to multiple nodes.

Given any valid algorithm A for chain online tracking, let $y_i(t)$ be the best knowledge of $f(t)$ at node n_i at any time instance t, for $i = 1, \ldots, h+1$. The first observation on a chain topology is given by the following lemma.

Lemma 4. *For an algorithm A (either online or offline) that solves the chain online tracking problem, we must have $y_i(t) \in [f(t) - \Delta, f(t) + \Delta]$ for any $i \in [1, h+1]$ in order to reduce communication while ensuring correctness. This holds for any $t \in [0, t_{now}]$.*

Proof. Consider any such algorithm A, the statement trivially holds for $i = h+1$, since node n_{h+1} is the tracker T and $g(t) = y_{h+1}(t)$. By the requirement of the chain online tracking problem, $g(t)$ must be within $[f(t)-\Delta, f(t)+\Delta]$ at any time instance $t \in [0, t_{\text{now}}]$.

Next, we show that at any time instance t, $y_i(t)$ must be within $[f(t) - \Delta, f(t) + \Delta]$ for any $i \in [1, h]$.

Assume that at some time instance $t \in [0, t_{now}]$, we have $|y_i(t) - f(t)| = \delta$ at node n_i and $\delta > \Delta$. Let ε_{i+1} be the tracking error threshold between n_i and n_{i+1} when running a tracking procedure between nodes n_i and n_{i+1}. Then, we must have $|y_{i+1}(t) - y_i(t)| \leq \varepsilon_{i+1}$. Hence, $|y_{i+1}(t) - f(t)| \leq \delta + \varepsilon_{i+1}$. Apply the same argument repeatedly at nodes $n_{i+1}, n_{i+2}, \ldots, n_{h+1}$, we can show that at n_{h+1}, it must be $|y_{h+1}(t) - f(t)| \leq \delta + \varepsilon_{i+1} + \cdots + \varepsilon_{h+1}$. That said, if $\delta > \Delta$, it could lead to $|y_{h+1}(t) - f(t)| = \delta > \Delta$, because the best (minimal) tracking errors for subsequent nodes in $n_{i+1}, \ldots, n_{h+1}$ are 0; they cannot be negative values.

Thus, the assumption that there exists a t and node n_i such that $|y_i(t) - f(t)| = \delta > \Delta$ must be wrong. □

Lemma 4 formalizes a very intuitive observation on a chain topology. This result helps us arrive at the following.

Lemma 5. *Both* CHAINTRACKA *and* CHAINTRACKR*'s competitive ratios are $+\infty$ for the chain online tracking problem.*

Proof. We prove the case for CHAINTRACKA. The proof for CHAINTRACKR is similar and omitted for brevity.

Consider a function f at the observer s (which is node n_0) whose values always change no more than Δ around a constant a. In other words, $f(t) \in [a-\Delta, a+\Delta]$ for any time instance t.

By the construction of CHAINTRACKA, we must have:
(1) $y_i(t) \in [f(t) - \frac{i}{h+1}\Delta, f(t) + \frac{i}{h+1}\Delta]$ for any $i \in [1, h]$;
(2) $g(t) = y_{h+1}(t) \in [f(t) - \Delta, f(t) + \Delta]$.

Consider an adversary Alice that tries to explore the worst case for CHAINTRACKA. Suppose that t_0 is the initial time instance. Alice first sets $f(t_0) = a - \Delta$. It takes $h+1$ messages to let n_{h+1} learn a valid value for $g(t_0)$ at time t_0. By the facts above, it must be $y_i(t_0) \in [a - \Delta - \frac{i}{h+1}\Delta, a - \Delta + \frac{i}{h+1}\Delta]$ for any $i \in [1, h]$.

Alice then sets $f(t_1) = a + \Delta$. Now $y_i(t_0)$ is more than Δ away from $f(t_1)$ for any $i \in [1, h]$. By Lemma 4, such $y_i(t_0)$'s are not allowed, hence, any node n_i cannot simply set $y_i(t_1) = y_i(t_0)$. Instead, every node n_i needs to receive an update message to produce a valid tracking value $y_i(t_1)$. This leads to h messages. Again, based on the design of CHAINTRACKA, $y_i(t_1) \in [a + \Delta - \frac{i}{h+1}\Delta, a + \Delta + \frac{i}{h+1}\Delta]$.

Alice sets $f(t_2) = a - \Delta$, by a similar argument, this will again trigger h messages. She repeatedly alternates the subsequent values for f between $a+\Delta$ and $a-\Delta$. CHAINTRACKA pays at least h messages for any $t \in [t_0, t_{now}]$, which leads to $O(ht_{now})$ messages in total. However, the offline optimal algorithm on this problem instance only needs to set $g(t_0) = y_{h+1}(t_0) = a$ at t_0, which takes $h+1$ messages, and keeps all subsequent $g(t_i)$ the same as $g(t_0)$.

Hence, cratio(CHAINTRACKA) $= ht_{now}/(h+1) = t_{now} = +\infty$. □

3.3.2 Optimal Chain Online Tracking

Recall that the centralized, two-party online tracking (simply known as online tracking) is a special case of chain online tracking with no relay nodes, i.e., $h = 0$. The OPTTRACK method in Algorithm 1 achieves an $O(\log \Delta)$-competitive ratio for the online tracking problem. Furthermore, it is also shown that $O(\log \Delta)$ is the lower bound for the competitive ratio of any online algorithms for online tracking [87, 89]. Yet, when generalizing it to chain online tracking with either CHAINTRACKA or CHAINTRACKR, the competitive ratio suddenly becomes unbounded. The huge gap motivates us to explore other alternatives,

which leads to the optimal chain online tracking method, CHAINTRACKO (chain-tracking-optimal).

This algorithm is shown in Algorithm 2, and its construction is surprisingly simple: allocate all error threshold to the very first relay node!

Basically, CHAINTRACKO ensures that $y_1(t)$ is always within $f(t) \pm \Delta$ using the OPTTRACK method. For any other relay node n_i for $i \in [2, h+1]$, it maintains $y_i(t) = y_{i-1}(t)$ at all time instances t. The tracker T maintains $g(t) = y_{h+1}(t)$ (recall node n_{h+1} is the tracker node). In other words, $g(t) = y_{h+1}(t) = y_h(t) = \cdots = y_2(t) = y_1(t)$ for any t.

Lemma 6. CHAINTRACKO*'s competitive ratio is* $O(\log \Delta)$ *for chain online tracking with* h *relay nodes.*

Proof. While running algorithm OPTTRACK between the observer s and node n_1, we define a round as a time period $[t_s, t_e]$, such that $S = [f(t_s) - \Delta, f(t_s) + \Delta]$ at t_s and $S = \emptyset$ at t_e from line 1 and line 2 in Algorithm 1. By the proof of Theorem 5 in [87, 89], we know that OPTTRACK will communicate $O(\log \Delta)$ messages in a round. Thus, by the construction of Algorithm 2, CHAINTRACKO has to communicate $O(h \log \Delta)$ messages in this round.

For a round $[t_s, t_e]$, consider any time instance $t \in [t_s, t_e]$. Lemma 4 means that $y_i(t) \in [f(t) - \Delta, f(t) + \Delta]$ for any $i \in [1, h+1]$ in the offline optimal algorithm. Suppose node n_i receives no message in this round, then it must be the case that:

$$y_i(x) \in \cap_{t=t_s}^{x} [f(t) - \Delta, f(t) + \Delta] \text{ for any } x \in [t_s, t_e]. \tag{3.1}$$

Consider the set $S(x)$ at node n_1 at time x, where $S(x)$ is the set S at time x in Algorithm 1. By the construction of Algorithm 1, $S(x) = \cap_t [f(t) - \Delta, f(t) + \Delta]$ for a subset of time instances t from $[t_s, x]$ (only when $|f(t) - g(t)| > \Delta, S \leftarrow S \cap [f(t) - \Delta, f(t) + \Delta]$).

Algorithm 2: CHAINTRACKO (Δ, h)

1 Let the tracking output at a node n_i be $y_i(t)$.
2 Run OPTTRACK (Δ) between observer s and the first relay node n_1, by running n_1 as a tracker.
3 **for** *any node* n_i *where* $i \in [1, h]$ **do**
4 Whenever $y_i(t)$ has changed, send $y_i(t)$ to node n_{i+1} and set $y_{i+1}(t) = y_i(t)$.

Clearly, we must have:

$$\cap_{t=t_s}^{x} [f(t) - \Delta, f(t) + \Delta] \subseteq S(x) \text{ for any } x \in [t_s, t_e]. \tag{3.2}$$

By the end of this round, $S(t_e)$ becomes $\emptyset$ by the definition of a round, which means that $\cap_{t=t_s}^{x}[f(t) - \Delta, f(t) + \Delta]$ has become $\emptyset$ at some time $x \leq t_e$ by (4.4). But this means that (4.2) cannot be true. Hence, our assumption that node n_i receives no message in this round must be wrong. This argument obviously applies to any node n_i for $i \in [1, h + 1]$, which implies that an offline optimal algorithm must have sent at least $h + 1$ messages in this round.

Thus, $\mathsf{cratio}(\textsc{ChainTrackO}) = ((h + 1) \log \Delta)/(h + 1) = \log \Delta$. □

CHAINTRACKO is optimal among the class of online algorithms that solve the chain online tracking problem, in terms of its competitive ratio. Specifically, $O(\log \Delta)$ is the lower bound for the competitive ratio of any online algorithms for chain online tracking. Let $\mathsf{C\text{-}OPT}(h)$ be the number of messages sent by the offline optimal algorithm for a chain online tracking problem with h relay nodes.

Lemma 7. *Any online algorithms for chain online tracking of h relay nodes must send $\Omega(\log \Delta \cdot \mathsf{C\text{-}OPT}(h))$ messages.*

Proof. Suppose A is an online algorithm for chain online tracking with h relay nodes. The approximations of $f(t)$ at different nodes are $y_1(t), y_2(t), \ldots, y_h(t), y_{h+1}(t)$, respectively (note that n_{h+1} is the tracker T, and $g(t) = y_{h+1}(t)$).

Consider an adversary Alice, who maintains an integer set R such that any $y \in R$ at time instance t is a valid representative for the value of $f(t)$, for $t \in [t_s, t_e]$. Note that a round is defined as a period $[t_s, t_e]$, such that $R = [f(t_s) - \Delta, f(t_s) + \Delta]$ at t_s and $R = \emptyset$ at t_e. Let d_i be the number of integers in R after the i-th update sent out by algorithm A from the observer. Initially, $R = [f(t_s) - \Delta, f(t_s) + \Delta]$ and $d_0 = 2\Delta + 1$.

In each round $[t_s, t_e]$, we will show that there exists a sequence of $f(t)$ values such that A has to send $\Omega(\log \Delta \cdot \mathsf{C\text{-}OPT}(h))$ messages, but the optimal offline method has to send only $\mathsf{C\text{-}OPT}(h)$ messages.

Consider a time instance $t \in [t_s, t_e]$ after the i-th update sent by algorithm A. Let

x be the median of R. Without loss of generality, suppose more than $\frac{h+1}{2}$ values among $\{y_1(t), y_2(t), \ldots, y_{h+1}(t)\}$ are in the range $[\min(R), x]$, let $Y_{\leq}(t)$ be the set of such values and $z = \max(Y_{\leq}(t))$.

Alice sets $f(t+1) = z + \Delta + 1$. It is easy to verify that $f(t+1) - y_i(t) > \Delta$ for any $y_i(t) \in Y_{\leq}(t)$. Hence, any node n_i, such that $y_i(t) \in Y_{\leq}(t)$, must receive an update at $(t+1)$ to ensure that $y_i(t+1) \in [f(t+1) - \Delta, f(t+1) + \Delta]$ by Lemma 4. This means that A has to send at least $\frac{h+1}{2}$ messages (the size of $Y_{\leq}(t)$) from t to $t+1$.

After the $(i+1)$-th update sent out by A from the observer, R has to be updated as $R \leftarrow R \cap [f(t+1) - \Delta, f(t+1) + \Delta]$, i.e., $R = [z+1, x + (d_i - 1)/2]$ and its size reduces by at most half at each iteration when A sends out an update from the observer. It is easy to see that $d_{i+1} \geq (d_i - 1)/2$. Using the same argument, we will get a similar result if more than $\frac{h+1}{2}$ values of $\{y_1(t), \ldots, y_{h+1}(t)\}$ are in the range $[x, \max(R)]$. In that case, we set $f(t+1) = z - \Delta - 1$ where $z = \min(Y_{\geq}(t))$.

That said, it takes at least $\Omega(\log \Delta)$ iterations for R to be a constant, since R's initial size is $O(\Delta)$ and its size reduces by at most half in each iteration. When R becomes empty, Alice starts a new round. By then, an offline optimal algorithm must have to send at least one update from the observer to the tracker (as $g(t)$ must hold a value from R to represent $f(t)$ for $t \in [t_s, t_e]$), which takes at least $O(h)$ messages. Hence, in any such round $[t_s, t_e]$, when $R = \emptyset$ at t_e, $\mathsf{C\text{-}OPT}(h) = O(h)$, but A has to send at least $\Omega(\frac{h+1}{2} \log \Delta)$ messages, which completes the proof. □

3.4 The Broom Case

A base case for distributed online tracking is the "broom" topology, as shown in Figure 3.1(a). A broom topology is an extension of the chain topology where there are m observers (instead of only one) connected to the first relay node. Similarly as before, we denote the ith relay node as n_i, and n_1 is the first relay node that connects directly to m observers. In fact, a broom topology reduces to a chain topology when $m = 1$.

Since there are m functions, one from each observer, an important distinction is that the function to be tracked is computed based on these m functions. Specifically, the goal is to track $f(t)$ where $f(t) = f(f_1(t), \ldots, f_m(t))$ for some function f at T subject to an error threshold Δ. Clearly, the design of online tracking algorithms in this case will

have to depend on the function f. We *focus on the max aggregate function* in this work, and discuss other aggregate functions in Section 3.6. Hence, in subsequent discussions, $f(t) = \max(f_1(t), \ldots, f_m(t))$ and $g(t)$ must be in the range $[f(t) - \Delta, f(t) + \Delta]$ at T, for any time instance t.

3.4.1 A Baseline Method

A baseline method is to ask T to track each function $f_i(t)$ within $f_i(t) \pm \Delta$ for $i \in [1, m]$ using a function $g_i(t)$. The tracker computes $g(t) = \max(g_1(t), \ldots, g_m(t))$ for any time instance t. For the ith function, this can be done by using the CHAINTRACKO method for a chain online tracking instance, where the chain is the path from observer s_i to tracker T. Given that $g_i(t) \in [f_i(t) - \Delta, f_i(t) + \Delta]$ for all $i \in [1, m]$, it is trivial to show that $g(t) \in [f(t) - \Delta, f(t) + \Delta]$. We denote this approach as the m-CHAIN method.

3.4.2 Improvement

Recall that CHAINTRACKO allocates all error threshold to the first relay node n_1 in its chain; all other relay nodes simply forward every update arrived at n_1 (from observer s) to T. Hence, in the m-CHAIN method, it is n_1 that actually tracks $g_1(t), \ldots, g_m(t)$ and n_1 simply passes every update received for $g_i(t)$ through the chain to T. This clearly generates excessive communication. In light of this, we consider a class A_{broom} of online algorithms for broom online tracking as follows:

1. Every node u in a broom topology keeps a value $y_u(t)$ which represents the knowledge of u about $f(t)$ in the subtree rooted at u at time t. For a leaf node u (an observer), $y_u(t)$ is simply its function value $f_u(t)$.
2. Each leaf node u's function is tracked by its parent v within error Δ using $g_u(t)$, i.e., $|g_u(t) - f_u(t)| \leq \Delta$ for every time instance t. Note that $g_u(t)$ does not need to be $f_u(t)$. Specifically, a leaf u sends a new value $g_u(t)$ to its parent v *only when* $|g_u(last) - f_u(t)| > \Delta$, where $g_u(last)$ is the previous update u sent to v.

Note that in *both broom and tree models*, we *do not* analyze the competitive ratio (cratio) of their online algorithms. The reason is that in a broom or a tree topology, since the offline optimal algorithm offline can see the entire input instance in advance, offline can "communicate" between leaf nodes for free. These are observers that are distributed in the online case. As a result of this, there always exists an input instance where the performance

gap between an online algorithm and offline is infinitely large.

Hence, in the following discussion, we will analyze the performance of an online algorithm using the concept of ratio as defined in Section 3.2.1 with respect to the class A_{broom}.

In a broom topology, we use $y_i(t)$ to denote $y_u(t)$ for a node u that is the ith relay node n_i.

Lemma 8. *Any algorithm $A \in A_{broom}$ must track functions $f_1(t), \ldots, f_m(t)$ with an error threshold that is exactly Δ at the first relay node n_1 in order to minimize* ratio(A).

Proof. Assume that the claim does not hold. This means there exists an algorithm $A \in A_{broom}$ that allows node n_1 to track at least one function $f_i(t)$ using an error threshold δ that is less than Δ. Without loss of generality, consider $i = 1$. In this case, we can show that ratio$(A) = +\infty$ by constructing an input instance I as follows.

In this instance I, $f_j(t) = -2\Delta$ for $j = 2, \ldots, m$ and any time instance t; $f_1(t) = (-1)^{t \mod 2}\Delta$. In other words, $f_1(t)$ alternates between $-\Delta$ and Δ and all other functions are set to a constant -2Δ. In this case, clearly, $f(t) = \max(f_1(t), \ldots, f_m(t)) = f_1(t)$ for all time instances t. But since the error threshold allocated to n_1 for f_1 is $\delta < \Delta$, A needs to send $+\infty$ number of messages when t goes $+\infty$, no matter how it designs its online tracking algorithm between n_1 and s_1.

But the optimal online algorithm *for this particular instance* only needs to send $(m + h + 1)$ number of messages in the first time instance. This algorithm A^*_I sets $g_1(t_1) = f_1(t_1) + \Delta$, and $g_i(t_1) = f_i(t_1)$ for all $i > 1$ at the first time instance t_1. It then asks each observer s_i to only send an update to n_1 if and only if at a time instance t, $f_i(t)$ has changed more than Δ from its last communicated value to n_1. At any time instance, n_1 sends $y_1(t) = \max(g_1(t), \ldots, g_m(t))$ to T, through the chain defined by $n_2, \ldots, n_h$, if $y_1(t)$ is different from the last communicated value from n_1 to T. It is easy to verify that this algorithm belongs to A_{broom} and it works correctly on all possible instances. On the above input instance, at the first time instance t_1, it will send $g_1(t_1) = 0$, and $g_2(t_1) = \cdots = g_m(t_m) = -2\Delta$ to n_1 from observers $s_1, \ldots, s_m$ (n_1 also forwards only $y_1(t) = 0$ to T). But it will incur no more communication in all subsequent time instances. Hence, its communication cost is $(m + h + 1)$.

Thus, on this input instance I, $\mathsf{ratio}(A, I)/\mathsf{ratio}(A^*_I, I) = +\infty$. As a result, $\mathsf{ratio}(A) = +\infty$.

Note that the optimal algorithm for a particular input instance may perform badly on other input instances. The definition of ratio is to quantify the difference in the worst case between the performance of an online algorithm A against the best possible performance for a valid input instance. □

Lemma 8 implies that $y_u(t)$ at every relay node u must be tracked by its parent node exactly, since all error thresholds have to be allocated to the first relay node n_1.

That said, whenever $y_i(t)$ changes, the message will be popped up along the chain to the tracker T. Formally,

Lemma 9. *Whenever $y_i(t) \neq y_i(t-1)$ at node n_i for any $i \in [1, h]$, any $A \in A_{broom}$ must send an update from n_i to n_{i+1}, and this update message must be $y_i(t)$.*

Proof. This lemma is an immediate result of the above discussion. □

Lemmas 8 and 9 do *not* imply that A_{broom} does not have many choices, because there are still many possible tracking strategies between n_1 and the m leaf nodes (observers). But these two lemmas do help us reach the following theorem.

Theorem 6. *For any algorithm A in A_{broom}, there exists an input instance I and another algorithm $A' \in A_{broom}$, such that $\mathsf{cost}(A, I)$ is at least h times worse than $\mathsf{cost}(A', I)$, i.e., for any $A \in A_{broom}$, $\mathsf{ratio}(A) = \Omega(h)$.*

Proof. For simplicity and without loss of generality, it suffices to prove the theorem for $m = 2$, i.e., a broom topology that has only 2 observers at the leaf level. We denote the two observers as s_1 and s_2, respectively. For an algorithm $A \in A_{broom}$, we will play as an adversary to construct two bad input instances I_1 and I_2 with respect to A.

Initially, we set $f_1(t_1) = \Delta$. Suppose in algorithm A s_1 sends a value x_1 to n_1. Note that x_1 must be an integer in $[0, 2\Delta]$. When $x_1 > 0$ we will construct input instance I_1; otherwise we will construct instance I_2 for $x_1 = 0$.

Table 3.1 shows the construction of I_1 and the behavior of A on I_1. At time t_1, we set $f_2(t_1) = x_1 + \frac{3}{2}\Delta$ and s_2 sends a value x_2 to n_1. Clearly, x_2 is strictly larger than

Table 3.1: Input instance I_1 and behavior of A.

Time instance	$f_1(t)$	$g_1(t)$	$f_2(t)$	$g_2(t)$	$y_1(t)$
Initialization					
t_1	Δ	x_1	$x_1 + \frac{3}{2}\Delta$	x_2	x_2
t_2	$x_1 + \Delta$	x_1	$\ll 0$	$\ll 0$	x_1
Round					
t_{2i+1}	$x_1 + \Delta$	x_1	$x_1 + \frac{3}{2}\Delta$	x_2	x_2
t_{2i+2}	$x_1 + \Delta$	x_1	$\ll 0$	$\ll 0$	x_1

x_1 since $x_2 \geq x_1 + \frac{\Delta}{2}$. Hence, at node n_1, $g_1(t_1) = x_1$ and $g_2(t_1) = x_2$. We have $y_1(t_1) = \max(g_1(t_1), g_2(t_1)) = x_2$ and it will be propagated all the way up to the tracker at the root node by Lemma 9.

At time t_2, we update $f_1(t_2) = x_1 + \Delta$ and $f_2(t_2)$ to a value that is $\ll 0$. Note that s_1 sends no message to n_1 since $|x_1 - f_1(t_2)| \leq \Delta$; meanwhile, s_2 needs to send an update message that is $\ll 0$ to n_1 because $|g_2(t_1) - f_2(t_2)| > \Delta$ and $f_2(t_2) \ll 0$. Now, $y_1(t_2)$ will be set back to x_1 which will be propagated up to the tracker T.

The rest of I_1 is constructed in rounds. Right before each round, we always ensure that $y_1(t) = x_1$ and $g_2(last) \ll 0$. Each round i contains two time instances t_{2i+1} and t_{2i+2}. In a round i, we keep f_1's value at $x_1 + \Delta$ and alternate the values of f_2 between $x_1 + \frac{3}{2}\Delta$ and some value that is $\ll 0$. Specifically, at time t_{2i+1}, s_2 will send a value x_2 to n_1 and $g_2(t_{2i+1})$ will be set to x_2. But s_1 will not send any message and $g_1(t_{2i+1})$ will still be x_1. Because $|x_2 - (x_1 + \frac{3}{2}\Delta)| \leq \Delta$, clearly, $x_2 > x_1$. Hence, $y_1(t_{2i+1}) = x_2$, and by Lemma 9, $y_1(t_{2i+1}) = x_2$ will be propagated up to the tracker. At the following time t_{2i+2}, s_2 sends another update message to n_1 that sets $g_2(t_{2i+2})$ to be $\ll 0$, and s_1 again sends no update message and $g_1(t_{2i+2})$ is still x_1. Hence, $y_1(t_{2i+2})$ goes back to x_1. Again, this update on $y_1(t)$ will be propogated up to the tracker according to Lemma 9.

In summary, for every time instance in a round, A will incur h messages. Hence, $\mathsf{cost}(A, I_1)$ equals $O(ht_{now})$.

Next, we show the existence of another algorithm A' in A_{broom} which only sends $O(h + t_{now})$ messages on the same input I_1.

At time t_1, A' sends $g_1(t_1) = 0$ from s_1 and $g_2(t_1) = x_1 + \Delta$ from s_2 to n_1, for $f_1(t_1) = \Delta$ and $f_2(t_1) = x_1 + \frac{3}{2}\Delta$, respectively. Hence, $y_1(t_1) = x_1 + \Delta$ and it will be

propagated up to the tracker at the root.

At time t_2, $f_1(t_2) = x_1 + \Delta$. This forces s_1 to send a new update message to n_1 because $|f_1(t_2) - g_1(t_1)| = |x_1 + \Delta - 0| > \Delta$ (note that while constructing I_1, we assumed that $x_1 > 0$). At this time, A' sends $g_1(t_2) = x_1 + \Delta$ for tracking $f_1(t_2) = x_1 + \Delta$. On s_2, $f_2(t_2) \ll 0$ which also forces an update message $g_2(t_2) \ll 0$ to be sent to n_1. But $y_1(t_2) = \max(g_1(t_2), g_2(t_2))$, which still equals $y_1(t_1) = x_1 + \Delta$! Thus, $y_1(t_2) = x_1 + \Delta$ does not need to sent up to the tracker along the chain.

In subsequent rounds, A' is able to maintain $y_1 = x_1 + \Delta$ with respect to I_1 in each round, as shown in Table 3.2. That said, it only takes two messages for updating $g_2(t)$ at n_1 in each round. Therefore, cost$(A', I_1) = h + 4 + 2 \times r$ for r rounds, which is $O(h + t_{now})$ since each round has two time instances. This means that $\mathsf{cost}(A', I_1) = O(h + t_{now})$.

Similarly, we can construct a bad input instance I_2 for algorithm A when $x_1 = 0$, as shown by Table 3.3. And for this input instance, there exists another algorithm A'' in A_{broom} that only takes $(h + 3 + 2 \times r) = O(h + t_{now})$ messages on input I_2, as shown by Table 3.4.

Table 3.2: A' on input instance I_1.

Time instance	$f_1(t)$	$g_1(t)$	$f_2(t)$	$g_2(t)$	$y_1(t)$
Initialization					
t_1	Δ	0	$x_1 + \frac{3}{2}\Delta$	$x_1 + \Delta$	$x_1 + \Delta$
t_2	$x_1 + \Delta$	$x_1 + \Delta$	$\ll 0$	$\ll 0$	$x_1 + \Delta$
Round					
t_{2i+1}	$x_1 + \Delta$	$x_1 + \Delta$	$x_1 + \frac{3}{2}\Delta$	$x_1 + \Delta$	$x_1 + \Delta$
t_{2i+2}	$x_1 + \Delta$	$x_1 + \Delta$	$\ll 0$	$\ll 0$	$x_1 + \Delta$

Table 3.3: A on input instance I_2.

Time instance	$f_1(t)$	$g_{s_1}(t)$	$f_2(t)$	$g_{s_2}(t)$	$y_1(t)$
Initialization					
t_1	Δ	0	$\Delta + 1$	x_2	x_2
t_2	Δ		$\ll 0$	$\ll 0$	0
Round					
t_{2i+1}	Δ		$\Delta + 1$	x_2	x_2
t_{2i+2}	Δ		$\ll 0$	$\ll 0$	0

Table 3.4: A'' on input instance I_2.

Time instance	$f_1(t)$	$g_{s_1}(t)$	$f_2(t)$	$g_{s_2}(t)$	$y_1(t)$
Initialization					
t_1	Δ	1	$\Delta+1$	1	1
t_2	Δ		$\ll 0$	$\ll 0$	1
Round					
t_{2i+1}	Δ		$\Delta+1$	1	1
t_{2i+2}	Δ		$\ll 0$	$\ll 0$	1

That said, for any algorithm $A \in A_{broom}$, there always exists an input instance I and an algorithm $A' \in A_{broom}$, such that $\mathsf{cost}(A, I) = O(ht_{now})$ and $\mathsf{cost}(A', I) = O(h + t_{now})$. In other words, $\mathsf{ratio}(A) = \Omega(h)$. □

Theorem 6 implies that there does not exist an "overall optimal" algorithm A in A_{broom} that always achieves the smallest cost on all input instances from $\mathcal{I}$. Such an algorithm A would imply $\mathsf{ratio}(A) = 1$, which contradicts the above.

Next, we present an online algorithm whose performance is close to the lower bound established by Theorem 6.

3.4.3 The BROOMTRACK Method

We design the BROOMTRACK algorithm in Algorithm 3. Similarly as before, n_{h+1} refers to the tracker T and $g(t) = y_{h+1}(t)$.

Algorithm 3: BROOMTRACK (Δ, m, h)

```
run m instances of OPTTRACK (Δ), one instance per pair (s_i, n_1). note that s_i is the
  observer at the ith leaf node and n_1, the first relay node, behaves as a tracker in
  OPTTRACK, for i ∈ [1, m];
let g_j(t) be the tracking result at n_1 at time t for f_j(t);
for any time instance t do
    if no update in any OPTTRACK instances at n_1 then set y_1(t) = y_1(t − 1) else
        set y_1(t) = max(g_1(t), g_2(t), ..., g_m(t));
    for i = 1, ..., h do
        if t = 0 or y_i(t)! = y_i(t − 1) then
            send y_i(t) to n_{i+1} and set y_{i+1}(t) = y_i(t);
        else set y_i(t) = y_i(t − 1)
```

The idea of Algorithm 3 is inspired by the same principle we explored in the design of CHAINTRACKO for chain online tracking, which is to ask the first relay node to do all the tracking, and the remaining relay nodes simply "relay" the updates sent out by n_1. Specifically, n_1 tracks each function f_i from observer s_i *with error threshold* Δ, and monitors the maximum value among these tracking results; n_1 takes this value as $y_1(t)$, his tracking result for $f(t)$. Other than the first time instance $t = 0$, only a change in this value, when $y_1(t) \neq y_1(t-1)$, will cause a communication through the entire chain, to send $y_1(t)$ to the tracker T and set $g(t) = y_1(t)$. Otherwise, every node in the chain, including the tracker, simply sets $y_u(t) = y_u(t-1)$ without incurring any communication in the chain (from n_1 to T).

The correctness of BROOMTRACK is obvious: $|g_i(t) - f_i(t)| \leq \Delta$ for any i and t. And, at the tracker T, for any time instance t, $g(t) = y_1(t)$ and $y_1(t) = \max(g_1(t), \ldots, g_m(t))$ immediately lead to $|g(t) - f(t)| \leq \Delta$, for $f(t) = \max(f_1(t), \ldots, f_m(t))$.

Theorem 7. *With respect to online algorithms in* A_{broom}, $\mathsf{ratio}(\text{BROOMTRACK}) < h \log \Delta$.

Proof. Given an input instance $I \in \mathcal{I}$, we denote M_i as the number of messages between s_i and n_1 for tracking function f_i up to t_{now} by algorithm BROOMTRACK. Thus, n_1 will receive $\sum_{i=1}^{m} M_i$ messages from $s_1, \ldots, s_m$. In the worst case, all of them get propagated up from n_1 to the root. So $\mathsf{cost}(\text{BROOMTRACK}, I) \leq h \sum_{i=1}^{m} M_i$.

On the other hand, for any algorithm A ($A \neq$ BROOMTRACK) in A_{broom}, it takes at least $\frac{M_i}{\log \Delta}$ messages between s_i and n_1 to track $f_i(t)$ on the input I by Theorem 5. Further, A needs to propagate at least one message through the chain at the first time instance. Thus, $\mathsf{cost}(A, I) \geq h + \frac{\sum_{i=1}^{m} M_i}{\log \Delta}$. Hence, for any $I \in \mathcal{I}$, the following holds:

$$\mathsf{ratio}(\text{BROOMTRACK}, I) \leq \frac{h \sum_{i=1}^{m} M_i}{h + \frac{\sum_{i=1}^{m} M_i}{\log \Delta}} < h \log \Delta.$$

Hence, $\mathsf{ratio}(\text{BROOMTRACK}) < h \log \Delta$. □

Similarly, we can show that m-CHAIN's ratio is $O(h \log \Delta)$ with respect to online algorithms in A_{broom}.

Corollary 3. $\mathsf{ratio}(m\text{-CHAIN}) = O(h \log \Delta)$.

3.5 The General Tree Case

In a general tree topology, every leaf node is still an observer, but it is no longer necessary for all leaf nodes to appear in the same level in the tree. Furthermore, they do not need to share a single chain to the tracker T. We still assume that there are m observers and the tracker T locates at the root. Similarly as before, every non-leaf node (except the root node) is considered a relay node. Using a simililar definition as that for the class A_{broom} in the broom model, we can define A_{tree} as the class of online algorithms for the tree online tracking problem.

A trivial case is shown in Figure 3.3(a). Any leaf node (an observer) is connected through a path to the tracker at the root, and no two paths share a common node except the root node. Every such path is a chain, hence, we can run the m-CHAIN method in this case. It is easy to show that m-CHAIN method is *an instance optimal* online method for this simple case, with respect to the class A_{tree}. In other words, its ratio is $O(1)$ with respect to A_{tree}. The following discussion excludes this trivial case.

So in the general case, an observer at a leaf node is still connected to the tracker through a single path. But a path may join another path at a non-root node u. We call such node u a "merging node". Let p_i be the ith path connecting observer s_i to T. A path p_i may join another path p_j at a merging node u, as illustrated in Figure 3.3(b). The common part of p_i and p_j is a chain from u to T, and is denoted as $p_{i,j}$ for any such i, j. Note that a path p_i may join with multiple, different paths at either one or more merging node(s), as shown in Figure 3.3(b). We use $(p - p')$ to denote the *subpath in a path* p that is *not a part of* another path p'.

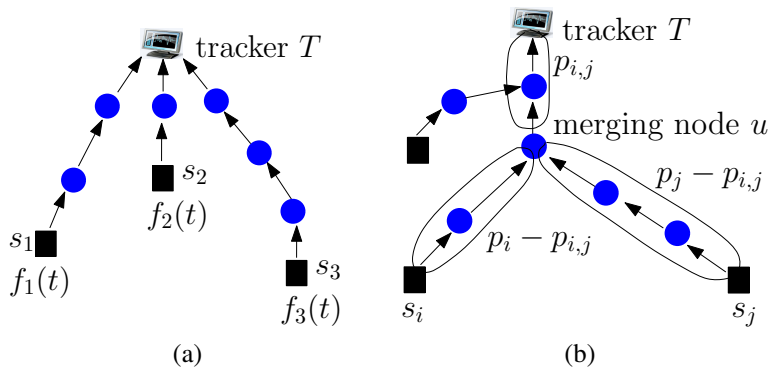


Figure 3.3: Tree online tracking. (a) simple tree. (b) general tree.

When two paths p_i and p_j share a merging node u, they form a *generalized broom model* in the sense that u connects to s_i and s_j through two separate chains, $(p_i - p_{i,j})$ and $(p_j - p_{i,j})$, respectively, and u itself connects to T through a single chain (that is $p_{i,j}$). When both s_i and s_j are directly connected under the merging node u, paths p_i and p_j become exactly a broom model (with two observers).

Given this observation, inspired by Theorem 6, we first have the following negative result.

Corollary 4. *There is no instance optimal algorithm for A_{tree}.*

Proof. Since we have excluded the trivial case in the tree model, a tree topology must have at least two paths p_i and p_j that share a merging node u. The paths p_i and p_j form a generalized broom model as discussed above. Suppose path $p_{i,j}$ consists of h relay nodes.

By Theorem 6, for any algorithm $A \in A_{tree}$, there always exists an input instance I and another algorithm $A' \in A_{tree}$, such that the cost of A on I on path $p_{i,j}$ is at least h times worse than the cost of A' on I on path $p_{i,j}$. The cost of A on I on path $(p_i - p_{i,j})$ and path $(p_j - p_{i,j})$ is at best the same as the cost of A' on I on these two paths. □

Also inspired by the above observation, we can extend the idea behind BROOMTRACK to derive the TREETRACK method for tree online tracking. It basically runs a similar version of BROOMTRACK on all generalized broom models found in a tree topology. This algorithm is shown below.

The correctness of TREETRACK is established by the following result.

Lemma 10. *Consider a node u. suppose $y_1, \ldots, y_\ell$ are the most recent updates of its ℓ child nodes. let z be the most recent update sent from u to its parent node. Define $y = \max\{y_1, \ldots, y_\ell\}$. If $y \neq z$, then u must send an update to its parent node, and this update message must be y.*

Proof. Without loss of generality, assume that $y = y_i$. The jth child node of u is referred to as the node j, and its function is denoted as f_j.

We prove the theorem by induction. First, consider the base case when u only has leaf nodes as its child nodes (i.e., u only has observers as its child nodes). We first show that u must send an update to its parent.

Algorithm 4: TREETRACK (Δ, a general tree R)

for *any non-leaf node u in R with an observer s_i as a leaf node directly connected under u* **do**
    run an instances of OPTTRACK (Δ) between u and s_i, where u is the tracker and s_i is the observer;
for *any non-leaf node u in R at any time instance* **do**
    let $y_1, \ldots, y_\ell$ be the most recent updates of its ℓ child nodes;
    let z be the most recent update of u to its parent;
    set $y = \max\{y_1, \ldots, y_\ell\}$;
    if $y \neq z$ **then**
        send y as an update to u's parent node in R
$g(t)$ at tracker T is the maximum value among the most recent updates T has received from all its child nodes.

case 1: $y > z$: For node i, set f_i to $y + \Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

case 2: $y < z$: For node i, set f_i to $y - \Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

With some technicality, we can show that in both cases: (1) $y \in [f(t) - \Delta, f(t) + \Delta]$ where $f(t) = \max(f_1(t), \ldots, f_\ell(t)))$; and (2) $z \notin [f(t) - \Delta, f(t) + \Delta]$. Hence, node u must send an update to its parent. Now suppose that node u sends an update y', but $y' \neq y$.

case 1: $y > y'$: For node i, set f_i to $y + \Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

case 2: $y < y'$: For node i, set f_i to $y - \Delta$. For node $j \neq i$, if $y_j \geq y - 2\Delta$, then set f_j to $y - \Delta$.

It is easy to show that y' is not in $[f_i(t) - \Delta, f_i(t) + \Delta]$, but by construction $f(t) = f_i(t)$. So the update cannot be such y'.

Now consider the case where u is a node such that the statement holds for all its descendants. We will show that the statement also holds for u. First, we show that u must send an update.

The fact that the statement holds for all the descendants of u implies that each y_j must be the most recent update of some leaf node in the jth subtree of u (rooted at u's jth child node). Let v be the leaf node corresponding to y_i.

case 1: $y > z$. For v, set f_v to $y + \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then

set f_w to $y - \Delta$.

case 2: $y < z$. For v, set f_v to $y - \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then set f_w to $y - \Delta$.

Using a similar argument as that in the base case, we can show that u must send an update to its parent. Now suppose that node u sends an update y', but $y' \neq y$.

case 1: $y > y'$. For v, set f_v to $y + \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then set f_w to $y - \Delta$.

case 2: $y < y'$. For v, set f_v to $y - \Delta$. For any leaf node $w \neq v$, if $y_w \geq y - 2\Delta$, then set f_w to $y - \Delta$.

Again by a similar argument as that in the base case, we can show that such y' cannot be the update message. □

Lastly, we denote h_{max} as the max length (the number of relay nodes in a path) of any path in a general tree.

Corollary 5. $\mathsf{ratio}(\textsc{TreeTrack}) = O(h_{max} \log \Delta)$ *with respect to* A_{tree}.

Proof. Given an input instance I and any algorithm $A \in A_{tree}$, suppose the number of messages between a leaf node (an observer) s_i and its parent node as M_i. It is easy to see that $\mathsf{cost}(\textsc{TreeTrack}, I) \leq h_{max} \sum_{i=1}^{m} M_i$. Meanwhile, any algorithm A' in A_{tree} satisfies $\mathsf{cost}(A', I) \geq h_{max} + \frac{\sum_{i=1}^{m} M_i}{\log \Delta}$ as a direct application of Theorem 5. Thus,

$$\mathsf{ratio}(\textsc{TreeTrack}, I) \leq \frac{h_{max} \sum_{i=1}^{m} M_i}{h_{max} + \sum_{i=1}^{m} M_i / \log \Delta} < h_{max} \log \Delta.$$

Therefore, $\mathsf{ratio}(\textsc{TreeTrack}) = O(h_{max} \log \Delta)$. □

3.6 Other Functions and Topologies

3.6.1 Other Functions for f

It is trivial to see that all of our results hold for $\min$ as well. That said, for any *distributive aggregates* on any topology, our methods can be extended to work for these aggregates. A *distributive aggregate* can be computed in a divide-and-conquer strategy, i.e., $f(f_1(t), \ldots, f_m(t)) = f(f(f_{a_1}(t), \ldots, f_{a_i}(t)), f(f_{a_{i+1}}(t), \ldots, f_{a_m}(t)))$, where $\{(a_1, \ldots, a_i), (a_{i+1}, \ldots, a_m)\}$ represents a (random) permutation and partition of $\{1, \ldots, m\}$. Other than

$\max$ and $\min$, another example is sum. In online tracking, since we assume that the number of observers is a constant, hence, tracking average is equivalent to tracking sum (because the count is a constant).

So consider sum as an example, we can extend m-CHAIN to both broom and tree models, by allocating Δ/m error for each chain. BROOMTRACK and TREETRACK can be extended as well, by: (1) allocating error thresholds *only* to a chain connecting to a merging node; (2) and making sure that the sum of error thresholds from *all chains to all merging nodes* equals Δ. The rest of the algorithm is designed in a similar fashion as that in BROOMTRACK and TREETRACK, respectively. It is easy to see that, no matter how many merging nodes there are, there are exactly m such chains. So a simple scheme is to simply allocate the error threshold Δ equally to any chain connecting to a merging node. Of course, the ratio of these algorithms needs to be analyzed with respect to the class of online algorithms for sum, which will be different from that in the max case. Such analyses are beyond the scope of this paper, and will be studied in the full version of this work.

More interestingly, our methods can be extended to work with *holistic aggregates* (aggregates that cannot be computed distributively) in certain case, such as *any quantiles* in a *broom* topology. Specifically, we can modify both m-CHAIN and BROOMTRACK for max to derive similar m-CHAIN and BROOMTRACK methods for distributed online tracking with any quantile function in a broom topology. Suppose $f(t) = \phi(f_1(t), \ldots, f_m(t))$, where $\phi(S)$ represents an ϕ-quantile from a set S of one-dimension values for any $\phi \in (0, 1)$. When $\phi = 0.5$, we get the median function.

The only change needed for m-CHAIN is to change $f = \max$ to $f = \phi$-quantile at the tracker, when applying f over $g_1(t), \ldots, g_m(t)$.

For BROOMTRACK, the only change we need to make is in line 6 in Algorithm 3, by replacing $\max$ with ϕ-quantile, i.e., $y_1(t) = \phi(g_1(t), \ldots, g_m(t))$. The following lemma ensures the correctness of these adaptions.

Lemma 11. *Let* $y_1(t) = \phi(g_1(t), \ldots, g_m(t))$ *and* $f(t) = \phi(f_1(t), \ldots, f_m(t))$. *If* $|g_i(t) - f_i(t)| \leq \Delta$ *for all* $i \in [1, m]$, *then it must be* $|y_1(t) - f(t)| \leq \Delta$.

Proof. We prove this with contradiction, and we illustrate the proof using median ($\phi = 0.5$). Other quantile functions can be similarly proved. To ease the discussion for quantiles,

we assume no duplicates. Cases with duplicates can be easily handled with a proper tie-breaker. Let us say $f(t) = f_i(t)$ for some i.

Suppose this is not true, then it must be $|y_1(t) - f_i(t)| > \Delta$. Without loss of generality, let us say $y_1(t) - f_i(t) > \Delta$. This means that $y_1(t) > g_i(t)$, since $|g_i(t) - f_i(t)| \leq \Delta$.

There are two cases. First, consider m is an odd number. Since $f_i(t)$ is the median of $f_1(t), \ldots, f_m(t)$, there must be $\frac{m-1}{2}$ functions $f_{\ell_1}, \ldots, f_{\ell_{(m-1)/2}}$ in $f_1(t), \ldots, f_m(t)$, such that $f_{\ell_i}(t) < f_i(t)$.

Consider $f_{\ell_i}(t)$ for any $i \in [1, (m-1)/2]$, the facts that $|g_{\ell_i}(t) - f_{\ell_i}(t)| \leq \Delta$, $f_{\ell_i}(t) < f_i(t)$ and $y_1(t) - f_i(t) > \Delta$ imply that $g_{\ell_i}(t) < y_1(t)$.

But now there are $(m+1)/2$ functions ($g_{\ell_1}, \ldots, g_{\ell_{(m-1)/2}}$, and $g_i(t)$), from $g_1(t), \ldots, g_m(t)$, that are less than $y_1(t)$, which contracts that $y_1(t)$ is the median of $g_1(t), \ldots, g_m(t)$.

The other case when m is an even number can be argued in the same fashion, as long as median is properly defined (as either the $(m-1)/2$th value or the $(m+1)/2$th value in the sorted sequence). □

Furthermore, using similar arguments, we can show a similar lower bound and upper bound, as that for the max case, on the ratio of these algorithms with respect to the class of online algorithms for tracking a quantile function in the broom model.

For the general-tree model, the m-CHAIN method still works for tracking any quantile function (since the tracker T tracks $f_1(t), \ldots, f_m(t)$ within error Δ with $g_1(t), \ldots, g_m(t)$). However, the TREETRACK method no longer works. The fundamental reason is that one cannot combine quantiles from two subtrees to obtain the quantile value of the union of the two subtrees. A similar argument holds against combining the tracking results from two subtrees in the case of quantile online tracking.

3.6.2 Other Topologies

As we already mentioned in Section 3.2, the general tree topology can be used to cover the cases when a relay node also serves as an observer at the same time. The idea is illustrated in Figure 3.4(a). A conceptual observer s' can be added as a leaf-level child node to such a relay node u. Our algorithms and results are carried over to this case. The only difference is that there is no need to run OPTTRACK between s' and u. Instead, u gets $g'(t) = f'(t)$ for free, where $f'(t)$ is the function at s' (the function at u when he acts as an

observer). This is useful when intermediate nodes in a network or a distributed system also observe data values of interest to the computation of f being tracked.

Lastly, our results from the general tree topology also extend to a graph. On a graph topology G, a distributed online tracking instance has a tracker T at a node from the graph, and m observers on m other nodes on the graph. We can find the shortest path from an observer s_i to T on the graph G, for each $i \in [1, m]$, where the length of a path is determined by the number of graph nodes it contains. Then, a general tree topology can be constructed (conceptually) from these m shortest paths by merging any common graph node from these paths into a single parent node with proper child nodes. T is the root node, and each observer is a leaf node. An example is shown in Figure 3.4(b). It can be shown that the two instances are equivalent in the context of distributed online tracking (when the communication is measured by the number of messages sent from one node to another). y

3.7 Experiment

All algorithms were implemented in C++. We simulated various distributed topologies, and executed all experiments in a Linux machine with an Intel Core I7-2600 3.4GHz CPU and 8GB memory. Note that every single communication between *two directly connected nodes* u and v contributes one message.

3.7.1 Datasets and Setup

We used two real datasets. The first dataset is a temperature dataset (TEMP) from a national atmosphere observation network. It contains temperature measurements from Jan 1997 to Oct 2011 from 26,383 distinct stations across the United States. We randomly select a subset of stations as the observers and treat readings from a station as the values of

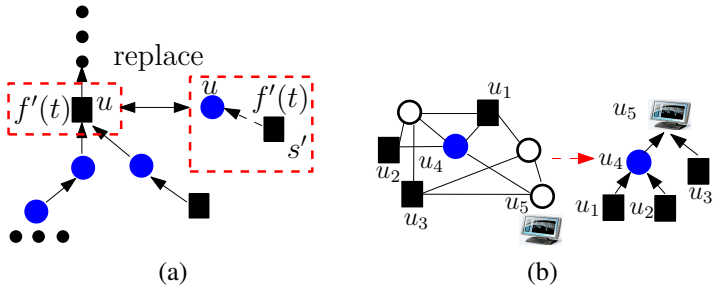


Figure 3.4: Other topologies. (a) observer at relay node. (b) graph topology.

the function for that observer.

The second dataset is wind direction measurements (WD) from a sea weather observation network. The wind direction measures the directional degree of the wind. Raw readings from research vessels were obtained which consist of approximately 11.8 million records observed during a 9-month interval in 2010. We partition the records into chunks, then randomly select a subset of chunks and treat the readings from each chunk as the values of an observer's function.

In all datasets, the readings are sorted by the time value they arrived. These two datasets provide quite different distributions. To illustrate this, we plot the function values of the function from an observer using a small sample (1000 time instances) in Figure 3.5.

We use TEMP as the default dataset. The default values of key parameters are: the number of time instances $N = 5000$; the number of relay nodes in a chain or a broom topology by default is $h = 2$. The default aggregate function f is $\max$. For any function f_i, we compute its standard deviation (std) with respect to $t \in [1, N]$. We set $\tau = \text{avg}(\text{std}(f_1), \ldots, \text{std}(f_m))$ for $f = \max$. We then set the default Δ value to 0.6τ. For the broom model, we set the default number of observers at $m = 15$, i.e., the number of leaf nodes connecting to the first relay node.

Note that leaf nodes can sit on different levels in a general tree topology. To produce a tree topology, each child node of an internal node with fanout F becomes a leaf node with probability p. We stop expanding nodes when they reach the tree level that equals the tree height H. We set $F = 3$, $p = 0.5$ and $H = 4$ as default values when generating a general tree topology.

For each experiment, we *vary the values for one parameter while setting the other*

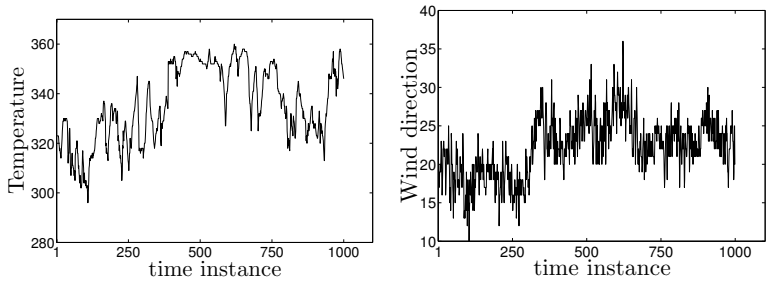


Figure 3.5: $f_1(t)$ for TEMP and WD, for $t \in [1, 1000]$. (a) TEMP. (b) WD.

parameters at their default values. We report the *average total number of messages per time instance* in a topology for different methods. This is denoted as "msgs per time instance" in our experimental results.

3.7.2 Chain Model

Figure 3.6(a) shows the communication cost when we vary Δ from 0.2τ to τ. Clearly, the communication cost reduces for all methods as Δ increases, since larger Δ values make a tracking algorithm less sensitive to the change of function values. CHAINTRACKO outperforms both CHAINTRACKA and CHAINTRACKR for all Δ values by an order of magnitude. Compared with the cost of the offline optimal method, denoted as offline, CHAINTRACKO performs well consistently. Averaging over the whole tracking period, offline needs 0.015 message per time instance and CHAINTRACKO takes only 0.056 message per time instance when $\Delta = 0.6\tau$.

Figure 3.6(b) shows the communication cost as h increases from 0 to 4. Note that when $h = 0$ the chain model becomes the centralized setting (one observer connects to the tracker directly). Not surprisingly, all methods need more messages on average as the chain contains more relay nodes while h increases. Among the three online algorithms, CHAINTRACKO gives the best performance for all h values. Meanwhile, we verified that its competitive ratio is indeed independent of h, by calculating the ratio between the number of messages sent by CHAINTRACKO and offline.

We then vary the number of time instances N from 1000 to 10,000 in Figure 3.6(c). We observe that the communication cost of all methods first decreases and then increases around $N = 5000$. This is explained by the dynamic nature of functions values over time, due to the real datasets we have used in our experiments.

Figure 3.6(d) shows the ratio between the cost of a method and the cost of offline, on both TEMP and WD datasets. Clearly, on both datasets, CHAINTRACKO has significantly outperformed both CHAINTRACKA and CHAINTRACKR. The cost of CHAINTRACKO is very close to the cost of offline.

3.7.3 Broom Model

Figure 3.7(a) shows the communication cost as we vary m the number of observers in a broom topology from 5 to 25. We see that BROOMTRACK outperforms m-CHAIN

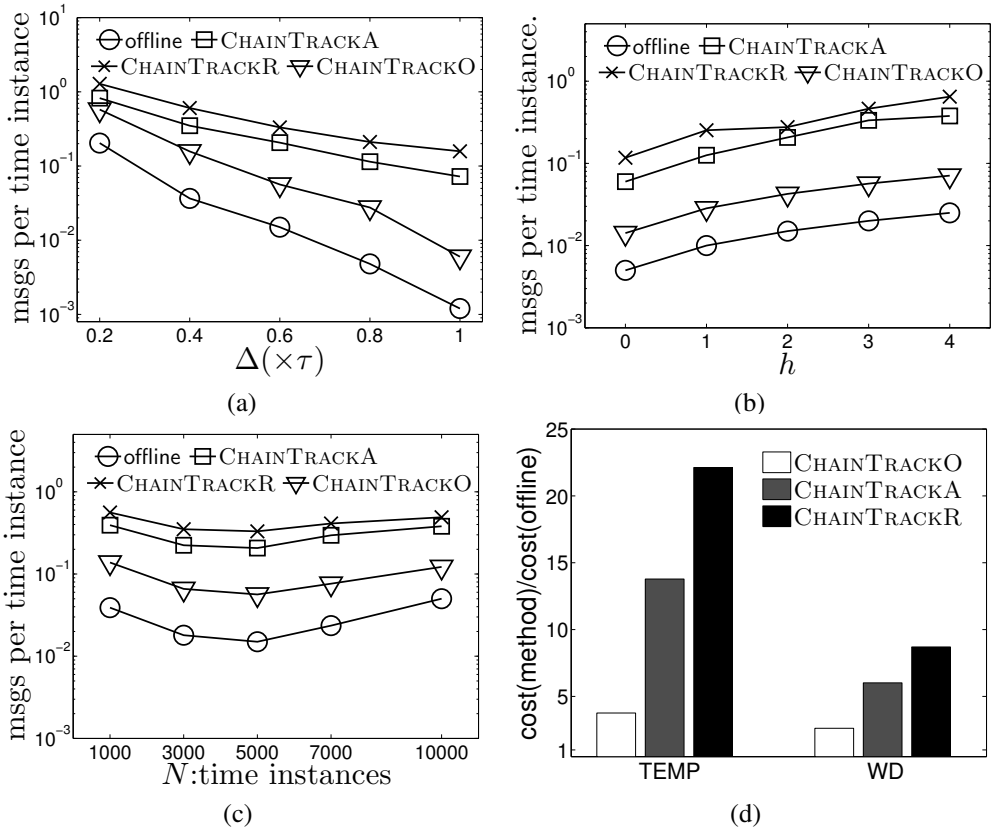


Figure 3.6: Performance of chain tracking methods on TEMP. (a) vary Δ. (b) vary h. (c) vary N. (d)cost(method)/cost(offline).

for all m values and the gap enlarges for larger m values. In particular, when $m = 15$ m-CHAIN takes on average 2.64 messages per time instance while BROOMTRACK takes only 1.04 messages per time instance, in a broom topology with 15 observers and 2 relay nodes. Note that if we were to track function values exactly, this means that we will need 45 messages per time instance!

In the following, we use $m = 15$ as the default value in a broom topology.

Figure 3.7(b) shows the communication cost when we change Δ from 0.2τ to τ. For all Δ values, BROOMTRACK has outperformed m-CHAIN by more than 3 times consistently. Again, for similar reasons, a larger Δ value always leads to less communication.

We change h, the number of relay nodes in a broom topology, from 0 to 4 in Figure 3.7(c). When $h = 0$, there is no relay node and all observers are directly connected to the tracker itself. Therefore, BROOMTRACK and m-CHAIN give the same communication cost when $h = 0$. We see that m-CHAIN suffers from the increase of h much more

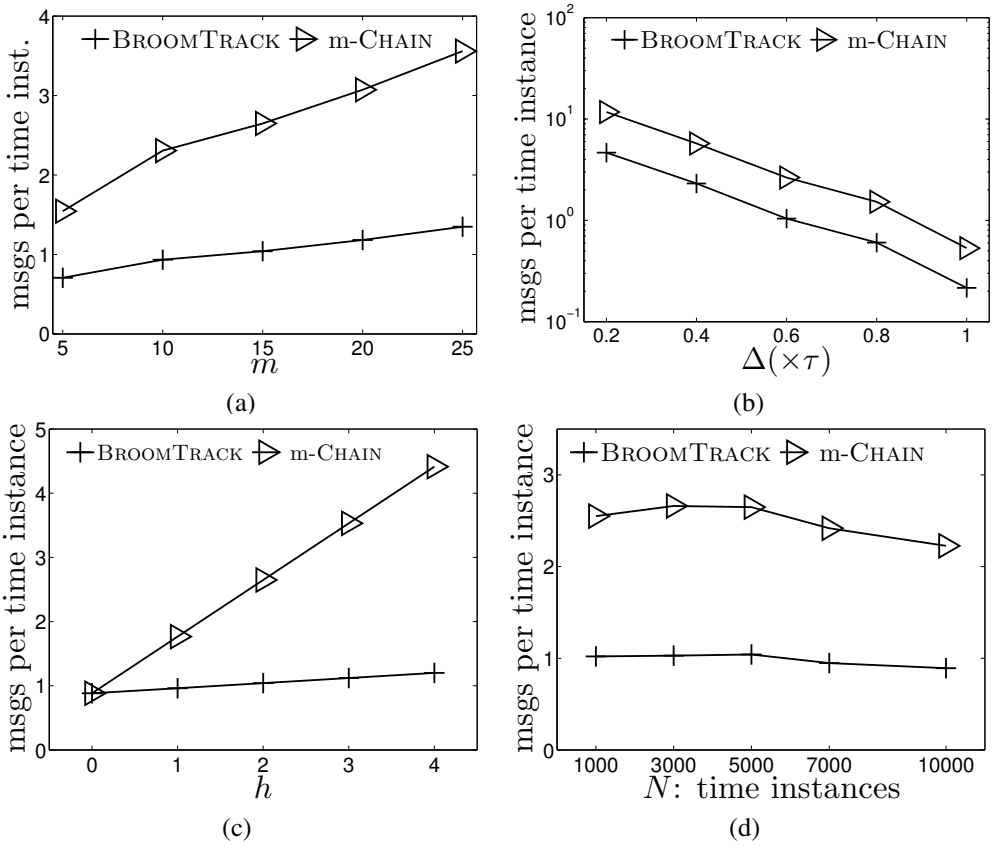


Figure 3.7: Performance of broom tracking methods on TEMP. (a) vary m. (b) vary Δ. (c) vary h. (d) vary N.

significantly compared to BROOMTRACK, and BROOMTRACK scales very well against more relay nodes. In particular, its number of messages per time instance only increases slightly from 0.88 to 1.20 when h goes from 0 to 4. Again, if we were to track all functions exactly, when the broom has 4 relay nodes and 15 observers, we will need 75 messages per time instance!

We vary N from 1000 to 10,000 in Figure 3.7(d). It shows that the average number of messages per time instance is quite stable and only decreases slightly when N goes beyond 5000 for both BROOMTRACK and m-CHAIN methods. This is caused by the change in the distribution of function values with respect to the time dimension.

Similar results were also observed on WD dataset, and they have been omitted for brevity.

3.7.4 General Tree Topology

We first vary p, when generating a tree topology, from 0.1 to 0.9 in Figure 3.8. The corresponding number of observers in the trees that were generated ranges from 27 to 7. Note that larger p values tend to produce less number of observers, since more nodes become leaf nodes (observers) early and they stop generating subtrees even though the height of the tree has not been reached yet in the tree generation process. Not surprisingly, the communication cost of both methods reduces as p increases on both datasets. Averaging over the whole tracking period, when $p = 0.5$ TREETRACK takes 1.28 messages per time instance while m-CHAIN takes 2.21 messages per time instance on TEMP dataset. This particular tree has 15 leaf nodes (observers) and 22 nodes in total. In the same tree topology, if we were to track function values exactly, we will need 41 messages per time instance! On WD dataset, both methods need even less number of messages per time instance, as shown in Figure 3.8(b). We set $p = 0.5$ as the default value in general-tree topologies.

Figure 3.9 shows the communication cost when we vary Δ from 0.2τ to τ. Again, TREETRACK outperforms m-CHAIN for all Δ values on both TEMP and WD datasets. In particular, these results show that TREETRACK is very effective in tracking changes of function values in a tree. For example, Figure 3.9(b) shows that TREETRACK takes on average 0.07 message per time instance when $\Delta = 0.6\tau$ on WD dataset.

Next, we grow the size of a tree by increasing H, the height of a tree, from 2 to 6 in Figure 3.10. Note that the number of nodes in a general tree increases exponentially in terms of H. Therefore, both methods show an increase in communication cost as H in-

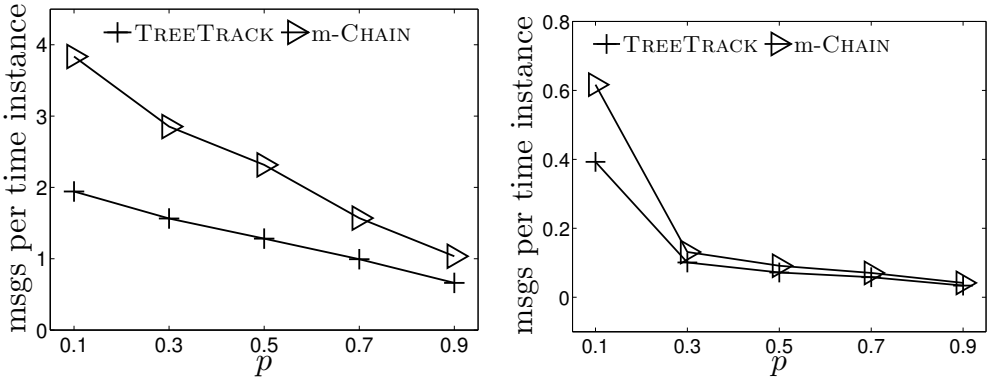


Figure 3.8: General-tree: vary p. (a) TEMP. (b) WD.

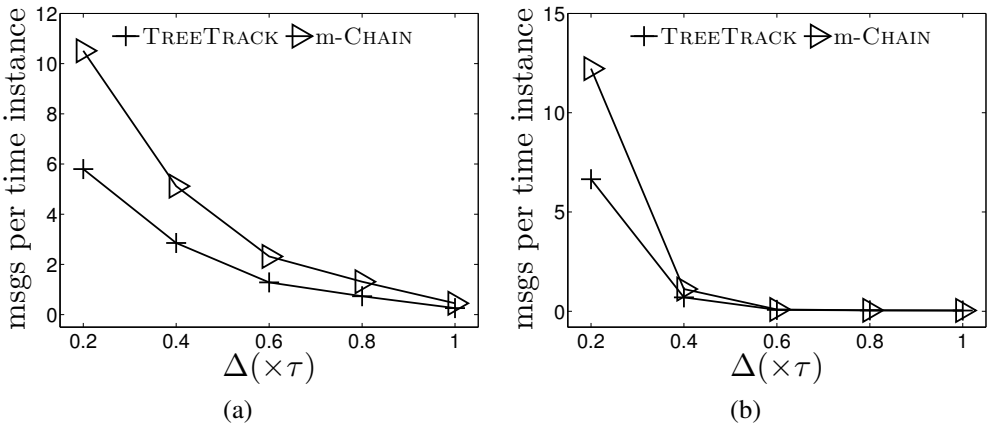


Figure 3.9: General-tree: vary Δ. (a) TEMP. (b) WD.

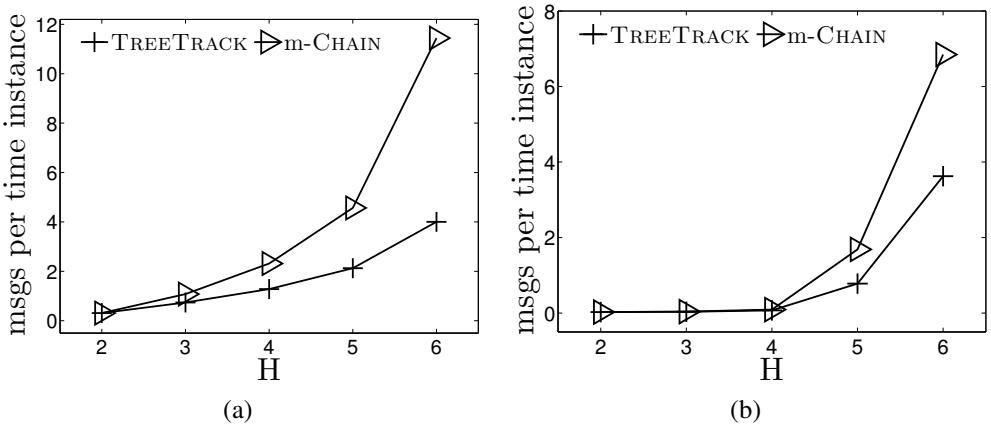


Figure 3.10: General-tree: vary H. (a) TEMP. (b) WD.

creases on both datasets. Nevertheless, we still see a significant performance improvement of TREETRACK over m-CHAIN as H increases on both WD and TEMP datasets. Also note that even though the communication cost increases as a tree grows higher, TREETRACK is still very effective. For example, when $H = 6$, we have a general tree of 91 nodes and 61 of them are leaf nodes (observers). Tracking these functions exactly would require more than 300 messages per time instance. But in this case, TREETRACK has sent on average only 3.6 messages and 4 messages on WD and TEMP datasets, respectively.

We vary the fan-out F from 2 to 4 in Figure 3.11. Not surprisingly, both m-CHAIN and TREETRACK show an increasing communication cost as F increases on both datasets since larger F values lead to more nodes in a tree. But the cost of TREETRACK increases much slowly. And in all cases, TREETRACK performs much better than m-CHAIN, and is

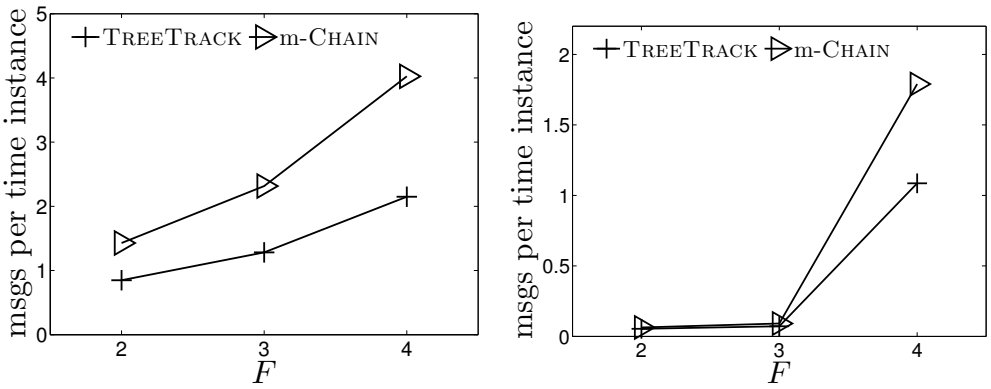


Figure 3.11: General-tree: vary F. (a) TEMP. (b) WD.

still orders of magnitude more effective if we compare its cost against the cost of tracking all functions exactly.

3.7.5 Other Functions

Lastly, we investigate our online algorithms for tracking sum and median aggregate functions, respectively. We evaluate their performance using the default parameter values on both datasets, over both broom and tree models. Note that under the default setting, if we were to tracking function values exactly, we will need 45 messages per time instance in the broom instance and 41 messages per time instance in the tree instance.

We first explore the performance of our methods for tracking sum function in Figure 3.12, for both broom and tree models. In this case, we define $\tau = \text{std}(f(t))$ for $t \in [1, N]$, where we calculate the values of $f(t)$ offline based on functions $f_1(t), \ldots, f_m(t)$, i.e., $f(t) = \sum_{i=1}^{m} f_i(t)$. Our improved methods (BROOMTRACK and TREETRACK) still outperform m-CHAIN in communication cost on both datasets.

Figure 3.13 compares the performance of different methods for tracking median function on both broom and tree models. Figure 3.13(a) shows that BROOMTRACK outperforms m-CHAIN in communication cost on both datasets. We evaluate the performance of m-CHAIN and m-CHAINA in Figure 3.13(b) for general-tree topology since TREETRACK does not work in this case. Here, m-CHAINA is a m-chain tracking method that calls CHAINTRACKA for each chain. It confirms our analysis that allocating tracking error to the first relay node indeed is better than allocating error threshold over different relay nodes

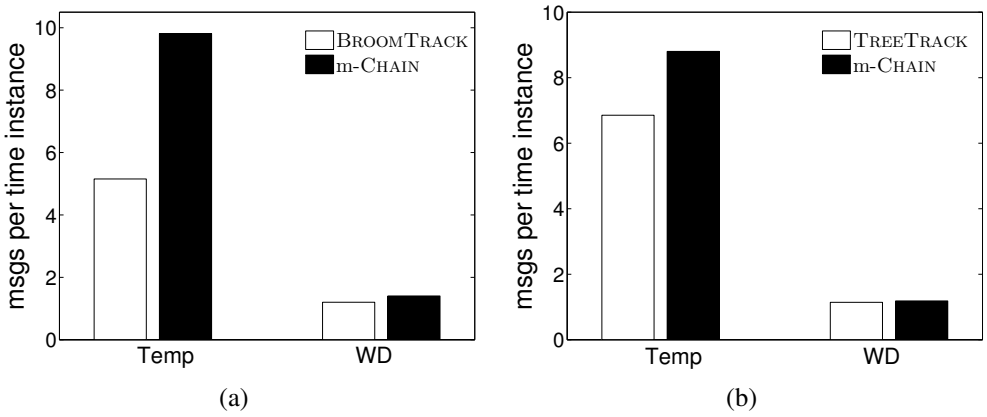


Figure 3.12: Track sum on broom and general tree. (a) Broom. (b) General-tree.

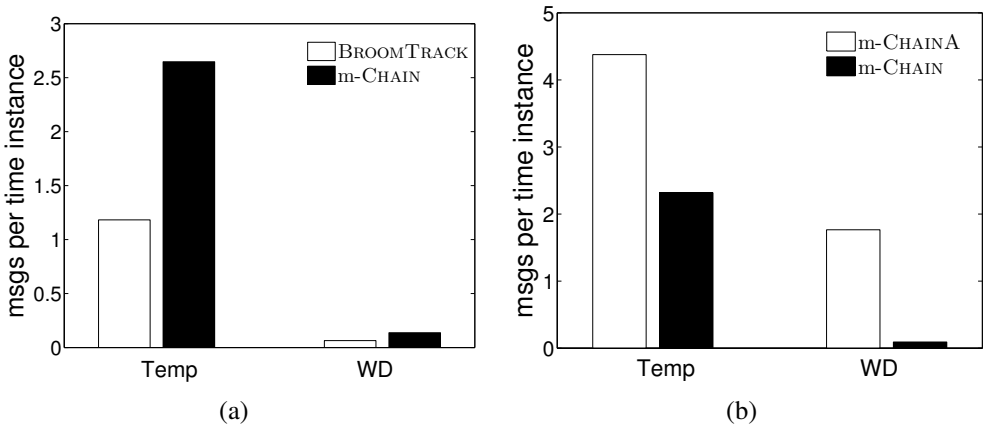


Figure 3.13: Track median on broom and general tree. (a) Broom. (b) General-tree.

in a chain, and m-CHAIN performs much better than m-CHAINA on both datasets.

3.8 Related Work

Online tracking is a problem that has only been recently studied in [87, 89], which we have reviewed in detail in Section 3.2. Only the centralized setting was studied in [87, 89].

In the popular *distributed streaming model*, the goal is to produce an approximation of certain functions/properties computed over the *union of data stream elements seen so far* for all observers, from the beginning of the time until now, for example, [15–17,40,88]). There are also variants in the distributed streaming model where a time-based sliding window of size > 1 with respect to t_{now} is used [20,21,56,64]). There are many works in the literature on tracking heavy hitters or quantiles over distributed streams that fall into this model, but

the heavy hitters or quantiles identified are computed with respect to the union of all data values from all observers between 0 and t_{now} or a sliding window [15–17, 40, 88], which is very different from our problem.

When f is the top-k function, a heuristic method has been proposed in [3] which provides no theoretical analysis. In contrast, our work focuses on common aggregation functions and principled methods with theoretical guarantees on their communication costs.

In distributed settings with multiple observers, to the best of our knowledge and as pointed out by the prior studies that have proposed the state-of-the-art centralized method [87, 89], a comprehensive study of the distributed online tracking problem with theoretical guarantees is still an open problem. Cormode et al. studied a special instance of this problem, but focused on only *monotone functions* [18, 19]. On the other hand, several existing studies explored the problem of threshold monitoring over distributed data [51, 53, 75], where the goal is to monitor continuously if $f(t)$ (a function value computed over the data values of all observers at time instance t) is above a user-defined threshold or not. Note that, threshold monitoring is different from the online tracking problem, where the tracker only needs to verify if the function value has exceeded a threshold or not, instead of keeping an approximation that is always within $f(t) \pm \Delta$. The geometric-based methods have been further explored to provide better solutions to the threshold monitoring problem [34], and the function approximation problem in the distributed streaming model [32] (which as analyzed above is different from the online tracking problem).

Lastly, our problem is different from distributed multiparty computation, which was explained in Section 4.1.

3.9 Conclusion

In this paper, we study the problem of distributed online tracking. We first extend the recent results for online tracking in the centralized, two-party model to the chain model, by introducing a number of relay nodes between the observer and the tracker. We then investigate both the broom model and the tree model, as well as other different tracking functions. Extensive experiments on real datasets demonstrate that our methods perform much better than baseline methods. Many interesting directions are open for future work, including but not limited to formally analyzing the ratios of our methods when extending

them to different aggregates, investigating online tracking with an error threshold that may change over time.

CHAPTER 4

SCALABLE HISTOGRAMS ON LARGE PROBABILISTIC DATA

4.1 Introduction

In many applications, uncertainty naturally exists in the data due to a variety of reasons. For instance, data integration and data cleaning systems produce fuzzy matches [28, 77]; sensor/RFID readings are inherently noisy [10, 25]. Numerous research efforts were devoted to represent and manage data with uncertainty in a probabilistic database management system [65, 77]. Many interesting mining problems have recently surfaced in the context of uncertain data, e.g., mining frequent pattern and frequent itemset [1, 5, 80]. In the era of big data, along with massive amounts of data from different application and science domains, uncertainty in the data is only expected to grow with larger scale.

Histograms are important tools to represent the distribution of feature(s) of interest (e.g., income values) [43, 67]. Not surprisingly, using the possible worlds semantics [23, 77], histograms are also useful tools in summarizing and working with probabilistic data [12–14]. Given that answering queries with respect to all possible worlds is in #P-complete complexity [23], obtaining a compact synopsis or summary of a probabilistic database is of the essence for understanding and working with large probabilistic data [12–14]. For example, they will be very useful for mining frequent patterns and itemsets from big uncertain data [1, 5, 80].

Cormode and Garofalakis were the first to extend the well-known V-optimal histogram (a form of bucketization over a set of one-dimension values) [43], and wavelet histogram [57] to probabilistic data [13, 14], followed by the work by Cormode and Deligiannakis [12]. Note that histogram construction can be an expensive operation, even for certain data, e.g., the exact algorithm for building a V-optimal histogram is based on a dynamic programming formulation, which runs in $O(Bn^2)$ for constructing B buckets over a domain

size of n [43]. Not surprisingly, building histograms on probabilistic data is even more challenging. Thus, existing methods [12–14] do not scale up to large probabilistic data, as evident from our analysis and experiments in this work.

Thus, this work investigates the problem of scaling up histogram constructions in large probabilistic data. Our goal is to explore quality-efficiency tradeoff, when such a tradeoff can be analyzed and bounded in a principal way. Another objective is to design methods that can run efficiently in parallel and distributed fashion, to further mitigate the scalability bottleneck using a cluster of commodity machines.

4.1.1 Overview

A probabilistic database characterizes a probability distribution of an exponential number of possible worlds, and each possible world is a realization (deterministic instance) of the probabilistic database. Meanwhile, the query result on a probabilistic database essentially determines a distribution of possible query answers across all possible worlds. Given the possible worlds semantics, especially for large probabilistic data, approximate query answering based on compact synopsis (e.g., histogram) is more desirable in many cases, e.g., cost estimations in optimizers and approximate frequent items [5, 12–14, 77, 80, 90].

Conventionally, histograms on a deterministic database seek to find a set of constant bucket representatives for the data distribution subject to a given space budget of buckets and an error metric. Building histograms on deterministic databases has been widely explored and understood in the literature. In probabilistic databases, building the corresponding histograms needs to address the following problems: (I) how to combine the histograms on each possible world; (II) how to compute the histogram efficiently without explicitly instantiating all possible worlds.

One meaningful attempt is building histograms that seek to minimize the expected error of a histogram's approximation of item frequencies across all possible worlds, using an error metric, which was first proposed in [13, 14]. One concrete application example might be estimating the expected result size of joining two probabilistic relations based on the corresponding histograms, or evaluating queries asking for an expected value approximately.

It is *important to note that for many error metrics, this histogram is not* the same as simply building a histogram for expected values of item frequencies; and the latter always

provides (much) worse quality in representing the probabilistic database with respect to a number of commonly used error metrics, as shown in [13, 14]

Based on this definition, a unified dynamic programming (DP) framework of computing optimal histograms on the probabilistic data was proposed in [13,14] with respect to various kinds of error metrics. Specifically, for the widely used sum of square error (SSE), it costs $O(Bn^2)$ time where B is the number of buckets and n is the domain size of the data. Immediately, we see that the optimal histogram construction suffers from quadratic complexity with respect to the domain size n. For a domain of merely $100,000$ values, this algorithm could take almost a day to finish and render it unsuitable for many datasets in practice.

Inspired by these observations, we propose constant-factor approximations for histograms on large probabilistic data. By allowing approximations, we show that it is possible to allow users to adjust the efficiency-quality tradeoff in a principal manner.

We summarize our contributions as follows. We propose a novel "partition-merge" method to achieve this objective. We introduce "recursive merging" to improve the efficiency, while the histogram quality achieved will not significantly deviate from the optimal version. We also devise novel synopsis techniques to enable distributed and parallel executions in a cluster of commodity machines, to further mitigate the scalability bottleneck. To that end,

- We review the problem of histogram constructions on probabilistic data in Section 4.2, and highlight the limitations in the state-of-the-art.
- We design PMERGE in Section 4.3, which gives constant-factor approximations and scales up the histogram construction on large probabilistic data. PMERGE uses a "partition-merge" approach to realize efficiency-quality tradeoff. It also admits "recursive-merging" to allow further efficiency-quality tradeoff.
- We extend our investigation to distributed and parallel settings in Section 4.4, and introduce novel synopsis methods to support computation- and communication-efficient execution of our methods in distributed and parallel fashion in Section 4.5.
- We conduct extensive experiments on large datasets in Section 4.6. The results suggest that our approximation methods have achieved significant (orders of magnitude) run-time improvement compared to the state-of-the-art approach with high-quality

approximation.

4.2 Background and State of the Art

4.2.1 Uncertain Data Models

Sarma *et al.* [72] describe various models of uncertainty, varying from the simplest *basic model* to the (very expensive) *complete model* that can describe any probability distribution of data instances.

The basic model is an over-simplification with no correlations. Existing work on histograms on uncertain data [12–14] adopted two popular models that extend the basic model, i.e., the *tuple model* and the *value model*, and compared their properties and descriptive abilities. The *tuple* and *value* models are two common extensions of the basic model in terms of the tuple- and attribute-level uncertainty [72], that were extensively used in the literature (see discussion in [12–14]).

Without loss of generality, we consider that a probabilistic database $\mathcal{D}$ contains one relation (table). We also concentrate on the one-dimension case or one attribute of interest.

Definition 2. The tuple model *was originally proposed in TRIO [2]. An uncertain database $\mathcal{D}$ has a set of tuples $\tau = \{t_j\}$. Each tuple t_j has a discrete probability distribution function (pdf) of the form $\langle (t_{j1}, p_{j1}), \ldots, (t_{j\ell_j}, p_{j\ell_j}) \rangle$, specifying a set of mutually exclusive* (item, probability) *pairs. Any t_{jk}, for $k \in [1, \ell_j]$, is* an item drawn from a fixed domain *and p_{jk} is the probability that t_j takes the value t_{jk} in the j^{th} row of a relation.*

When instantiating this uncertain relation to a possible world W, each tuple t_j either draws a value t_{jk} with probability p_{jk} or generates no item with probability of $1 - \sum_{k=1}^{\ell_j} p_{jk}$. The probability of a possible world W is simply the multiplication of the relevant probabilities.

Definition 3. The value model *is a sequence τ of independent tuples. Each tuple gives the* frequency distribution of a distinct item *of the form $\langle\ j : f_j = ((f_{j1}, p_{j1}), \ldots, (f_{j\ell_j}, p_{j\ell_j}))\ \rangle$. Here, j is an item drawn from a fixed domain (e.g., source IP) and its associated pdf f_j describes the distribution of j's possible frequency values.*

In particular, $\Pr[f_j = f_{jk}] = p_{jk}$ where f_{jk} is a frequency value from a frequency value domain $\mathcal{V}$; f_j is subject to the constraint that $\sum_{jk} p_{jk} \leq 1$ for $k \in [1, \ell_j]$. When it is

less than 1*, the remaining probability corresponds that* the item's frequency is zero. *When instantiating this uncertain relation to a possible world* W*, for an item* j*, its frequency* f_j *either takes a frequency value* f_{jk} *with probability* p_{jk} *or takes zero as its frequency value with probability* $1 - \sum_{k=1}^{\ell_j} p_{jk}$*. So the probability of a possible world* W *is computed as the multiplication of the possibilities of* f_j*'s taking the corresponding frequency in each tuple.*

4.2.2 Histograms on Probabilistic Data

Without loss of generality, in both models, we consider the items are drawn from the integer domain $[n] = \{1, ..., n\}$ and use $\mathcal{W}$ to represent the set of all possible worlds. Let N be the size of a probabilistic database, i.e., $N = |\tau|$.

For an item $i \in [n]$, g_i *is a random variable for the distribution of* i*'s frequency over all possible worlds*, i.e,

$$g_i = \{(g_i(W), \Pr(W)) | W \in \mathcal{W}\}, \tag{4.1}$$

where $g_i(W)$ is item i's frequency in a possible world W and $\Pr(W)$ is the possibility of W.

Example 3. *Consider an ordered domain* $[n]$ *with three items* $\{1, 2, 3\}$ *for both models, i.e.,* $n = 3$*.*

The input $\tau = \{\langle(1, \frac{1}{2}), (3, \frac{1}{3})\rangle, \langle(2, \frac{1}{4}), (3, \frac{1}{2})\rangle\}$ in the tuple model *defines eight possible worlds in Table 4.1. In contrast, the input* $\tau = \{\langle 1 : (1, \frac{1}{2})\rangle, \langle 2 : (1, \frac{1}{3})\rangle, \langle 3 : ((1, \frac{1}{2}), (2, \frac{1}{2}))\rangle\}$ in the value model *defines eight possible worlds in Table 4.2.*

Consider the *tuple model* example from above and denote the eight possible worlds (from left to right) as $W_1, \ldots, W_8$. It is easy to see that $g_3(W) = 1$ for $W \in \{W_4, W_6, W_7\}$,

Table 4.1: Example for tuple model

W	$\emptyset$	1	2	3	1, 2	1, 3	2, 3	3, 3
$\Pr(W)$	$\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{24}$	$\frac{1}{6}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{12}$	$\frac{1}{6}$

Table 4.2: Example for value model

W	3	1, 3	2, 3	3, 3	1, 2, 3	1, 3, 3	2, 3, 3	1, 2, 3, 3
$\Pr(W)$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$	$\frac{1}{12}$

$g_3(W) = 2$ for $W \in \{W_8\}$, and $g_3(W) = 0$ on the rest. Thus, the frequency random variable g_3 of item 3 is $g_3 = \{(0, \frac{1}{3}), (1, \frac{1}{2}), (2, \frac{1}{6})\}$ with respect to $\mathcal{W}$ in this example. Meanwhile, it is also easy to see $g_3 = \{(1, \frac{1}{2}), (2, \frac{1}{2})\}$ over $\mathcal{W}$ in the value model example from above.

Definition 4. *A* B-bucket representation *partitions domain* $[n]$ *into* B *nonoverlapping consecutive buckets* (s_k, e_k) *for* $k \in [1, B]$*, where* $s_1 = 1$*,* $e_B = n$ *and* $s_{k+1} = e_k + 1$*. Frequencies within each bucket* b_k *are approximated by a single representative* $\widehat{b}_k$ *and we represent it as* $b_k = (s_k, e_k, \widehat{b}_k)$*.*

The B-bucket histogram achieving the minimal SSE error for approximating a deterministic data distribution is known as the V-optimal histogram [42]. It can be found using a dynamic programming formulation in $O(Bn^2)$ time [43], where n is the domain size of the underlying data distribution. We denote this method from [43] as the OPTVHIST method.

To extend histogram definitions to probabilistic data, we first consider a single possible world $W \in \mathcal{W}$ for a probabilistic dataset $\mathcal{D}$, where W is *a deterministic dataset*. Hence, the frequency vector of W is given by $G(W) = \{g_1(W), ..., g_n(W)\}$ (recall that $g_i(W)$ is item i's frequency in W). Given a B-bucket representation for approximating $G(W)$, the SSE of a bucket b_k in the world W is given as: $SSE(b_k, W) = \sum_{j=s_k}^{e_k} (g_j(W) - \widehat{b}_k)^2$. The SSE of the B-bucket representation in W is simply $\sum_{k=1}^{B} SSE(b_k, W)$.

Cormode and Garofalakis have extended the B-bucket histogram to probabilistic data [13, 14] by asking for the minimal expected SSE. Formally,

Definition 5. *Given the (uncertain) frequency sequence of random variables* $\{g_1, \ldots, g_n\}$ *as defined in* (4.1)*, the problem seeks to construct a* B*-bucket representation (typically* $B \ll n$*) such that the expected SSE over all possible worlds is minimized, i.e., the histogram with the value given by:*

$$\mathcal{H}(n, B) = \min\{\mathrm{E}_{\mathcal{W}} \left[\sum_{k=1}^{B} \sum_{j=s_k}^{e_k} (g_j - \widehat{b}_k)^2 \right]\} \tag{4.2}$$

In (4.2), the expectation of the sum of bucket errors is equal to the sum of expectations of bucket errors [13, 14], i.e.,

$$\mathrm{E}_{\mathcal{W}}\left[\sum_{k=1}^{B}\sum_{j=s_k}^{e_k}(g_j-\widehat{b}_k)^2\right]=\sum_{k=1}^{B}\mathrm{E}_{\mathcal{W}}\left[\sum_{j=s_k}^{e_k}(g_j-\widehat{b}_k)^2\right]. \tag{4.3}$$

Consequently, the optimal histogram could be derived by a dynamic programming formulation as follows:

$$\mathcal{H}(i,j)=\min_{1\le\ell<i}\mathcal{H}(\ell,j-1)+\min_{\widehat{b}}(\ell+1,i,\widehat{b}), \tag{4.4}$$

where $\mathcal{H}(i,j)$ represents the minimal error from the optimal j-buckets histogram on interval $[1,i]$; $\min_{\widehat{b}}(\ell+1,i,\widehat{b})$ is the minimal bucket error for the bucket spanning the interval $[\ell+1,i]$ using a single representative value $\widehat{b}$.

Previous work [13, 14] showed that the cost of the optimal histogram is $O(Bn^2)$ and $\min_{\widehat{b}}(\ell+1,i,\widehat{b})$ could be computed in constant time using several precomputed prefix-sum arrays which we will describe in the following subsection. We dub this state-of-art method from [14] *the* OPTHIST *method*.

4.2.3 Efficient Computation of Bucket Error

Cormode and Garofalakis [14] show that, for SSE, the minimal error of a bucket $b=(s,e,\widehat{b})$ is achieved by setting the representative $\widehat{b}=\frac{1}{e-s+1}\mathrm{E}_{\mathcal{W}}\left[\sum_{i=s}^{e}g_i\right]$. The corresponding bucket error is given by:

$$SSE(s,e,\widehat{b})=\sum_{i=s}^{e}\mathrm{E}_{\mathcal{W}}[g_i^2]-\frac{1}{e-s+1}\mathrm{E}_{\mathcal{W}}[\sum_{i=s}^{e}g_i]^2. \tag{4.5}$$

In order to answer the $\min_{\widehat{b}}(s,e,\widehat{b})$ query in (4.4) for any (s,e) values in constant time, prefix-sum arrays of $\mathrm{E}_{\mathcal{W}}[g_i^2]$ and $\mathrm{E}_{\mathcal{W}}[g_i]$ in equation (4.5) are precomputed as follows (details can be found in [14]):

$$A[e]=\sum_{i=1}^{e}\mathrm{E}_{\mathcal{W}}[g_i^2]=\sum_{i=1}^{e}(\mathrm{Var}_{\mathcal{W}}[g_i]+\mathrm{E}_{\mathcal{W}}[g_i]^2)\quad B[e]=\sum_{i=1}^{e}\mathrm{E}_{\mathcal{W}}[g_i] \tag{4.6}$$

Tuple model: $\mathrm{E}_{\mathcal{W}}[g_i]=\sum_{t_j\in\tau}\Pr[t_j=i]$ and $\mathrm{Var}_{\mathcal{W}}[g_i]=\sum_{t_j\in\tau}\Pr[t_j=i](1-\Pr[t_j=i])$.

Value model: $\mathrm{E}_{\mathcal{W}}[g_i]=\sum_{v_j\in\mathcal{V}}v_j\Pr[g_i=v_j]$ and $\mathrm{Var}_{\mathcal{W}}[g_i]=\sum_{v_j\in\mathcal{V}}(v_j-\mathrm{E}_{\mathcal{W}}[g_i])^2\Pr[g_i=v_j]$

Set $A[0]=B[0]=0$, then the minimal SSE $\min_{\widehat{b}}(s,e,\widehat{b})$ for both models is computed

as:

$$A[e] - A[s-1] - \frac{(B[e] - B[s-1])^2}{e - s + 1}.$$

In both models, in addition to the $O(Bn^2)$ cost as shown in the last subsection, it also takes $O(N)$ cost to compute the A, B arrays ($N = |\tau|$, number of probabilistic tuples).

4.3 Approximate Histograms

The state-of-the-art OPTHIST method from [14] is clearly not scalable, when given larger domain size.

4.3.1 A Baseline Method

A natural choice is to consider computing a B-bucket histogram for the *expected frequencies of all items*. Note that this histogram is not the same as the desired histogram as defined in equation (4.2) and (4.3) since in general $\mathrm{E}[f(X)]$ does not equal $f(\mathrm{E}[X])$ for arbitrary function f and random variable X.

However, we can show in our histogram, the SSE error of a bucket $[s, e]$ using $\widehat{b}$ as its representative is:

$$SSE(s, e, \widehat{b}) = \mathrm{E}_{\mathcal{W}}[\sum_{i=s}^{e}(g_i - \widehat{b})^2] = \sum_{i=s}^{e}(\mathrm{E}_{\mathcal{W}}[g_i^2] - 2\mathrm{E}_{\mathcal{W}}[g_i]\widehat{b} + \widehat{b}^2).$$

On the other hand, if we build a B-bucket histogram over the expected frequencies of all items, the error of a bucket $[s, e]$ using $\bar{b}$ as its representative is:

$$SSE(s, e, \bar{b}) = \sum_{i=s}^{e}(\mathrm{E}_{\mathcal{W}}[g_i] - \bar{b})^2 = \sum_{i=s}^{e}((\mathrm{E}_{\mathcal{W}}[g_i])^2 - 2\mathrm{E}_{\mathcal{W}}[g_i]\bar{b} + \bar{b}^2).$$

When using the same bucket configurations (i.e., the same boundaries and $\widehat{b} = \bar{b}$ for every bucket), the two histograms above differ by $\sum_{j=s}^{e}(\mathrm{E}_{\mathcal{W}}[g_i^2] - (\mathrm{E}_{\mathcal{W}}[g_i])^2) = \sum_{j=s}^{e} \mathrm{Var}_{\mathcal{W}}[g_i]$ on a bucket $[s, e]$. Hence, the overall errors of the two histograms differ by $\sum_{i\in[n]} \mathrm{Var}_{\mathcal{W}}[g_i]$ which is a constant. Given this and computing the expected frequencies of all items can be done in $O(N)$ time, computing the optimal B-bucket histogram for them (now a deterministic frequency vector) still requires the OPTVHIST method from [43], taking $O(Bn^2)$ for a domain of size n, which still suffers the same scalability issue.

A natural choice is then to use an approximation for the B-bucket histogram on expected frequencies (essentially a V-optimal histogram), as an approximation for our histogram. The best approximation for a V-optimal histogram is an $(1+\varepsilon)$-approximation [79] (in fact, to the best of our knowledge, it is the only method with theoretical bound on approximation quality). But when using approximations, one cannot guarantee that the same bucket configurations will yield the same approximation bound with respect to both histograms. So its theoretical guarantee is no longer valid with respect to our histogram. Nevertheless, it is worth comparing to this approach as a baseline method, which is denoted as the *EF-Histogram* method.

4.3.2 The PMERGE Method

Hence, we search for novel approximations that can provide error guarantees on the approximation quality and also offer quality-efficiency tradeoff, for the histograms from [13, 14] as defined in (4.2). To that end, we propose a constant approximation scheme, PMERGE, by leveraging a "partition-merge" principle. It has a *partition phase* and a *merge phase*.

4.3.2.1 Partition

The *partition phase* partitions the domain $[n]$ into m equally-sized sub-domains, $[s_1, e_1]$, . . . , $[s_m, e_m]$ where $s_1 = 1, e_m = n$ and $s_{k+1} = e_k + 1$. For the kth sub-domain $[s_k, e_k]$, we compute the A, B arrays on this domain as A_k, B_k for $k \in [1, m]$. A_k and B_k are computed using $[s_k, e_k]$ as an input domain and equation (4.6) for the value and the tuple models, respectively,

Next, for each sub-domain $[s_k, e_k]$ ($k \in [1, m]$), we apply the OPTHIST method from [14] (as reviewed in Section 4.2.2) over the A_k, B_k arrays to find the *local optimal B-buckets histogram* for the kth sub-domain. The partition phase produces m local optimal B-bucket histograms, which lead to mB buckets in total.

4.3.2.2 Merge

The goal of the merge phase is to merge the mB buckets from the partition phase into optimal B buckets in terms of the SSE error using one merging step. To solve this problem, naively, we can view an input bucket $b = (s, e, \widehat{b})$ as having $(e - s + 1)$ items

with identical frequency value $\widehat{b}$. Then, our problem reduces to precisely constructing an V-optimal histogram instance [43]. But the cost will be $O(B(\sum_{i=1}^{mB}(e_i - s_i + 1))^2)$ using the OPTVHIST method, which is simply $O(Bn^2)$.

A critical observation is that a bucket $b = (s, e, \widehat{b})$ can also be viewed as a *single weighted frequency* $\widehat{b}$ with a weight of $(e - s + 1)$, such that we can effectively reduce the domain size while maintaining the same semantics. Formally, let $Y = mB$. A weighted frequency vector $\{f_1, f_2, \ldots, f_Y\}$ on an ordered domain $[Y]$ has a weight w_i for each f_i. It implies w_i items with a frequency f_i at i. The weighted version of the V-optimal histogram seeks to construct a B-bucket histogram such that the SSE between these buckets and the input weighted frequency vector is minimized. This problem is the same as finding:

$$\mathcal{H}_w(Y, B) = \min\{\sum_{k=1}^{B}\sum_{j=s_k}^{e_k} w_j(f_j - \widehat{b}_k)^2\},$$

where $s_1 = 1$ and $e_B = Y$. The optimal B buckets can be derived by a similar dynamic programming formulation as that shown in equation (4.4). The main challenge is to compute the optimal one-bucket $\min_{\widehat{b}}(s, e, \widehat{b})$ for any interval $[s, e]$ now in the weighted case.

4.3.2.3 Fast Computation of Bucket Error

We can show that in the weighted case the $\min_{\widehat{b}}(s, e, \widehat{b})$ is achieved by setting $\widehat{b} = \frac{\sum_{k=s}^{e} w_k f_k}{\sum_{k=s}^{e} w_k}$ and the corresponding bucket error for the bucket b is as follows: $SSE(b, \widehat{b}) = \sum_{j=s}^{e} w_j(f_j^2 - \widehat{b}^2)$. The prefix sum arrays that need to be precomputed are:

$$P[e] = \sum_{i=1}^{e} w_j f_j, \quad PP[e] = \sum_{i=1}^{e} w_j f_j^2, \quad W[e] = \sum_{i=1}^{e} w_j.$$

Given these arrays, $\min_{\widehat{b}}(s, e, \widehat{b})$ is computed as:

$$PP[e] - PP[s] - \frac{(P[e] - P[s])^2}{W[e] - W[s]}.$$

Thus, the weighted optimal B-bucket histogram can be derived by filling a $Y \times B$ matrix, and each cell (i, j) takes $O(Y)$ time. Thus, the weighted B-bucket histogram is computed in $O(BY^2) = O(m^2B^3)$ time, which is much less than $O(Bn^2)$ since both B and m are much smaller than n.

An example of PMERGE is given in Figure 4.1, where $n = 16$, $B = 2$, and $m = 4$. To

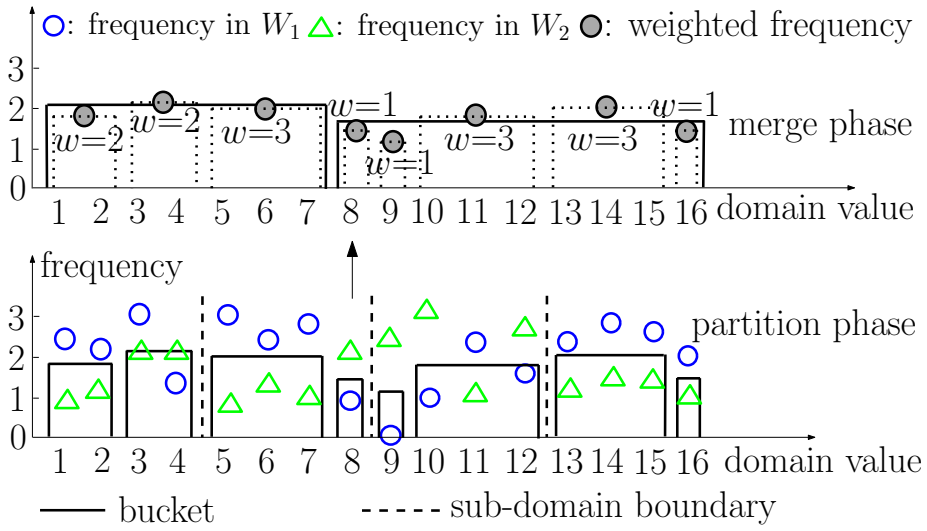


Figure 4.1: An example of PMERGE: $n = 16, m = 4, B = 2$.

ensure clarity, we show only two possible worlds W_1 (blue circle) and W_2 (green triangle) from the set of possible worlds $\mathcal{W}$ of this database. In the partition phase, each sub-domain of size 4 is approximated by 2 local optimal buckets. In total, the partition phase has produced 8 buckets in Figure 4.1. In the merge phase, each input bucket maps to a weighted frequency as discussed above. For example, the first bucket covering frequencies in $[1, 2]$ represents a weighted frequency of 1.8 with weight 2. These 8 buckets were merged into two buckets as the final output.

4.3.2.4 Complexity Analysis

In the partition phase, it takes linear time to compute the corresponding A_k, B_k arrays within each sub-domain $[s_k, e_k]$ for $k \in [1, m]$, following the results from [14]. The size of sub-domain $[s_k, e_k]$ is roughly n/m for $k \in [1, m]$. It takes $O(Bn^2/m^2)$ to run the OPTHIST method on A_k, B_k to find the kth local optimal B-bucket histogram. Next, the merge phase takes only $O(B^3m^2)$ time as analyzed above. Hence, with m sub-domains and one merging step, the following result immediately follows:

Lemma 12. PMERGE *takes* $O(N + Bn^2/m + B^3m^2)$.

4.3.2.5 Approximation Quality

In order to evaluate the absolute value of the histogram approximation error, we adopt the ℓ_2 distance (square root of SSE error) between the data distribution and the histogram

synopsis. Next, we show the approximation quality of PMERGE compared to the *optimal* B-bucket histogram found by OPTHIST in terms of the ℓ_2 distance.

Theorem 8. *Let* $\|\mathcal{H}(n,B)\|_2$ *and* $\|\mathcal{H}_{\text{PMERGE}}(n,B)\|_2$ *be the* ℓ_2 *norm of the SSE error of* B*-bucket histogram produced by* OPTHIST *and* PMERGE*, respectively, on domain* $[n]$*. Then,* $\|\mathcal{H}_{\text{PMERGE}}(n,B)\|_2 < 3.17 \cdot \|\mathcal{H}(n,B)\|_2$.

We denote the *probabilistic frequency vector* as $F = \{g_1, ...g_n\}$ (from either the tuple or the value model). Let $\widehat{g}_i$ denote the representative of g_i assigned by buckets from the OPTHIST method. Let $\bar{g}_i$ and $\widetilde{g}_i$ be the representative of g_i given by the buckets from the partition phase and the merge phase, respectively, in PMERGE. By definition, we have:

$$\mathcal{H}(n,B) = \sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \widehat{g}_i)^2 \Pr(W).$$

Similarly, the histogram error for PMERGE is

$$\mathcal{H}_{\text{PMERGE}}(n,B) = \sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \widetilde{g}_i)^2 \Pr(W).$$

By the optimality of the weighted histogram in the merge phase we must have

$$\sum_{i=1}^{n} (\bar{g}_i - \widetilde{g}_i)^2 \leq \sum_{i=1}^{n} (\bar{g}_i - \widehat{g}_i)^2. \tag{4.7}$$

Next, for any sub-domain $[s_k, e_k]$ ($k \in [1,m]$) in the partition phase, the optimality of the OPTHIST method [13, 14] ensures that PMERGE always produces the *optimal* B*-buckets histogram* for the probabilistic frequency vector $F_k = \{g_{s_k}, g_{s_k+1}, \ldots, g_{e_k}\}$. On the other hand, there are at most B buckets falling into $[s_k, e_k]$ to approximate F_k for the *optimal* B*-buckets histogram* produced by running the OPTHIST method on the entire F. Thus, we have

$$\sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \bar{g}_i)^2 \Pr(W) \leq \sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \widehat{g}_i)^2. \tag{4.8}$$

Finally, we also have:

Lemma 13. *For any* $a, b, c \in \mathbb{R}$*,*

$$(a-b)^2 \le 2(a-c)^2 + 2(c-b)^2. \tag{4.9}$$

Combining equations (4.7), (4.8), and (4.9) leads to:

$$\begin{aligned}
&\|\mathcal{H}_{\text{PMERGE}}(n,B)\|_2^2 = \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(g_i(W)-\widetilde{g}_i)^2\Pr(W)\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(2((g_i(W)-\bar{g}_i)^2+(\bar{g}_i-\widetilde{g}_i)^2))\Pr(W) \text{ by (4.9)}\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(2((g_i(W)-\bar{g}_i)^2+(\bar{g}_i-\widehat{g}_i)^2))\Pr(W) \text{ by (4.7)}\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(2(g_i(W)-\bar{g}_i)^2+4(\bar{g}_i-g_i(W)^2\\
& +4(g_i(W)-\widehat{g}_i)^2)\Pr(W) \qquad \text{by (4.9)}\\
=\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(6(g_i(W)-\bar{g}_i)^2+4(g_i(W)-\widehat{g}_i)^2)\Pr(W)\\
\le\ & 10\sum_{W\in\mathcal{W}}\sum_{i=1}^{n}d(g_i(W),\widehat{g}_i)\Pr(W) \qquad \text{by (4.8)}\\
=\ & 10\cdot\|\mathcal{H}(n,B)\|_2^2.
\end{aligned}$$

Hence, $\|\mathcal{H}_{\text{PMERGE}}(n,B)\|_2 < 3.17\cdot\|\mathcal{H}(n,B)\|_2$.

4.3.3 Recursive PMERGE

Note that the problem size of mB in the merge phase of PMERGE may still be too large to be handled efficiently by a DP method. Fortunately, we can further improve the efficiency by doing "recursive merging" as follows.

First of all, the partition phase will partition the input domain into m^ℓ equal-sized sub-domains, instead of only m sub-domains, for some integer ℓ (user specified).

The merge phase now *recursively* merges the $m^\ell B$ buckets from the partition phase into B buckets using ℓ iterations. Each iteration reduces the number of input buckets by a factor of m by applying a sequence of *merging steps*. Specifically, each merging step merges mB consecutive buckets (from left to right) from the current iteration into B buckets in the next iteration, which is done using the same merging step from the standard PMERGE method (i.e., using the weighted B-bucket histogram idea). We dub the recursive PMERGE methods RPMERGE.

Extending the analysis from Lemma 12 and Theorem 8 gives the following result, w.r.t the ℓ_2 norm of the SSE:

Theorem 9. *Using* $O(N + B\frac{n^2}{m^\ell} + B^3 \sum_{i=1}^{\ell} m^{(i+1)})$ *time, the* RPMERGE *method gives a* 3.17^ℓ *approximation of the optimal* B*-bucket histogram found by* OPTHIST.

We prove this by induction. When $\ell = 1$, this translates to the basic PMERGE method and the result from Theorem 8 implies the base case directly. Now assuming the theorem holds for some $\ell > 1$, We can show that the result also holds for $(\ell + 1)$. The derivation is similar to the machinery used in proving Theorem 8, albeit subjecting to some technicalities, we omit the details. The running time analysis follows directly from its construction.

To that end, we denote $\widehat{g}_i$ and $\widetilde{g}_i$ as the representative of g_i assigned by the OPTHIST method and the recursive version of PMERGE of depth $\ell + 1$. We have:

$$\|\mathcal{H}(n, B)\|_2^2 = \sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \widehat{g}_i)^2 \Pr(W).$$

Similarly, the histogram error for depth $(\ell + 1)$ PMERGE is

$$\mathcal{H}_{\text{RPMERGE}}(n, B) = \sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \widetilde{g}_i)^2 \Pr(W).$$

For the $(\ell + 1)$th merge phase, it calls m instances of depth ℓ PMERGE methods, each with a sub-domain of size n/m. Thus, we also use $\bar{g}_i$ and λ_i as the representative of g_i assigned by the depth ℓ PMERGE and the OPTHIST methods on such sub-domains. By induction assumption,

$$\sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \bar{g}_i)^2 \leq 10^\ell \sum_{W \in \mathcal{W}} \sum_{i=1}^{n} (g_i(W) - \lambda_i)^2. \tag{4.10}$$

By the optimality of the weighted DP formulation in the $(\ell + 1)$th merge phase we must have:

$$\sum_{i=1}^{n} (\bar{g}_i - \widetilde{g}_i) \leq \sum_{i=1}^{n} (\bar{g}_i - \widehat{g}_i). \tag{4.11}$$

Also, by the optimality of the OPTHIST method on each size n/m sub-domain we must have

$$\sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(g_i(W)-\lambda_i)^2 \le \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(g_i(W)-\widehat{g}_i)^2. \tag{4.12}$$

Combining equations (4.9), (4.10), (4.11), and (4.12) leads to:

$$\begin{aligned}
&\|\mathcal{H}_{\text{RPMERGE}}(n,B)\|_2^2 = \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(g_i(W)-\widetilde{g}_i)^2\Pr(W)\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}2((g_i(W)-\bar{g}_i)^2+(\bar{g}_i-\widetilde{g}_i)^2)\Pr(W) \text{ by (4.9)}\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}2((g_i(W)-\bar{g}_i)^2+(\bar{g}_i-\widehat{g}_i)^2)\Pr(W) \text{ by (4.11)}\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(2(g_i(W)-\bar{g}_i)^2+4(\bar{g}_i-g_i(W))^2\\
+\ & 4(g_i(W)-\widehat{g}_i)^2)\Pr(W) \text{ by (4.9)}\\
=\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(6(g_i(W)-\bar{g}_i)^2+4(g_i(W)-\widehat{g}_i)^2)\Pr(W)\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(6\cdot 10^{\ell}(g_i(W)-\lambda_i)^2\\
+\ & 4(g_i(W)-\widehat{g}_i)^2)\Pr(W) \text{ by (4.10)}\\
\le\ & \sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(6\cdot 10^{\ell}(g_i(W)-\widehat{g}_i)^2\\
+\ & 4(g_i(W)-\widehat{g}_i)^2)\Pr(W) \text{ by (4.12)}\\
=\ & (6\cdot 10^{\ell}+4)\sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(g_i(W)-\widehat{g}_i)^2\Pr(W)\\
\le\ & 10^{(\ell+1)}\sum_{W\in\mathcal{W}}\sum_{i=1}^{n}(g_i(W)-\widehat{g}_i)^2\Pr(W) = 10^{(\ell+1)}\|\mathcal{H}(n,B)\|_2^2.
\end{aligned}$$

Hence, for ℓ levels, $\|\mathcal{H}_{\text{PMERGE}}(n,B)\|_2 < 3.17^{\ell}\cdot\|\mathcal{H}(n,B)\|_2$.

It is important to note that the approximation bounds in both Theorems 8 and 9 reflect the *worst-case analysis*. The extreme cases leading to the worst-case bounds are almost impossible in real datasets. In practice, PMERGE and its recursive version RPMERGE always provide *(very) close to optimal approximation quality* (much better than what these worst-case bounds indicate), as shown in our experiments.

4.4 Distributed and Parallel PMERGE

PMERGE allows efficient execution in a distributed and parallel framework. In the partition phase, each sub-domain can be handled independently in parallel.

The recursive PMERGE offers even more venues for parallelism. In this case, *its merge phase* can also run in a distributed and parallel fashion, since each merging step from every iteration can be processed independently.

Next, we will address the challenge of computing the local A_k, B_k arrays efficiently for each sub-domain $[s_k, e_k]$ in a distributed and parallel setting . For both models, we assume that the underlying probabilistic database has been split into β chunks $\{\tau_1, \ldots, \tau_\beta\}$ and stored in a distributed file system (DFS). It is important to note that the input data are not necessarily sorted by the values of the items when stored into chunks in a DFS.

4.4.1 The Partition Phase in the Value Model

Recall that in the value model, f_i is a pdf describing item i's possible frequency values and their associated probabilities. We first show that:

Lemma 14. *In the value model,* $\Pr[g_i = v] = \Pr[f_i = v]$ *for any frequency value* $v \in \mathcal{V}$ *($\mathcal{V}$ is the domain of all possible frequency values).*

Let $\mathcal{W}_{i,v}$ be the set of possible worlds in which $g_i(W) = v$ (recall $g_i(W)$ is item i's frequency in W). Clearly, we have $\Pr(g_i = v) = \sum_{W \in \mathcal{W}_{i,v}} \Pr[W]$.

Next, let $\tau' = \tau - t_i$ (τ is the set of tuples forming the database, and $t_i = \{i, f_i\}$ is the ith tuple describing the possible frequency values for item i), and $\mathcal{W}_{\tau'}$ be the set of all possible worlds instantiated from τ'. Clearly, for any $W \in \mathcal{W}_{i,v}$, $\Pr[W] = \Pr[W' \text{ in } \mathcal{W}_{\tau'}] \cdot \Pr[f_i = v]$ where $W' = W - \{\text{all items } i\}$. Hence,

$$\begin{aligned} \Pr(g_i = v) &= \sum_{W \in \mathcal{W}_{i,v}} \Pr[W] \\ &= \sum_{W' \in \mathcal{W}_{\tau'}} \Pr[W'] \cdot \Pr[f_i = v] = \Pr[f_i = v]. \end{aligned}$$

Lemma 14 and equation (4.6) imply that:

Lemma 15. *The A, B arrays for the value model also equal:* $A[j] = \sum_{i=1}^{j} \mathrm{E}[f_i^2]$, $B[j] = \sum_{i=1}^{j} \mathrm{E}[f_i]$.

The ith tuple $t_i = (i, f_i = \{f_{i1}, \ldots, f_{i\ell_i}\})$ of a value model tuple set τ specifies the possible frequency values of item i. When instantiating a possible world W from τ, $g_i(W)$ must be in the set $\{f_{i1}, \ldots, f_{i\ell_i}\}$ or take a frequency of zero. Given this and Lemma 14, we have

$$\begin{aligned} \mathrm{E}_W[g_i] &= \sum_{W \in \mathcal{W}} g_i(W) \Pr(W) \\ &= \sum_{v \in \mathcal{V}} v \Pr[g_i = v] = \sum_{j=1}^{\ell_i} f_{ij} \Pr[g_i = f_{ij}] \\ &= \sum_{j=1}^{\ell_i} f_{ij} \Pr[f_i = f_{ij}] = \mathrm{E}[f_i]. \end{aligned}$$

Similarly, we can show $E_{\mathcal{W}}[g_i^2] = \mathrm{E}[f_i^2]$. Hence, A, B arrays in equation (4.6) for the value model can be rewritten as:

$$A[e] = \sum_{i=1}^{e} \mathrm{E}[f_i^2], \quad B[e] = \sum_{i=1}^{e} \mathrm{E}[f_i].$$

Without loss of generality, we assume β "data nodes (aka processes)" to consume the input data chunks, and also m "aggregate nodes/processes" to produce the local optimal B-bucket histograms. Each data chunk is processed by one data node in parallel. Each data node produces m partitions, each of which corresponds to a sub-domain of size (roughly) n/m, using a partition function $h : [n] \to [m]$, $h(i) = (\lceil i / \lceil n/m \rceil \rceil)$.

The ℓth data node processing chunk τ_ℓ reads in tuples in τ_ℓ in a streaming fashion. For each incoming tuple (i, f_i) found in τ_ℓ, it computes two values $(\mathrm{E}[f_i], \mathrm{E}[f_i^2])$. It then writes a key-value pair $(i, (\mathrm{E}[f_i], \mathrm{E}[f_i^2]))$ to the $h(i)$th partition. The $h(i)$th aggregate node will collect the $h(i)$th partitions from all β data nodes, the union of which forms the $h(i)$th sub-domain of the entire data.

Thus, the kth ($k \in [1, m]$) aggregate node will have all the key-value pairs $(i, (\mathrm{E}[f_i], \mathrm{E}[f_i^2]))$ for all $i \in [s_k, e_k]$ in the kth sub-domain, if item i exists in the database; otherwise, it simply produces a $(i, (0, 0))$ pair for such $i \in [s_k, e_k]$.

That said, the kth aggregate node can easily compute the A_k, B_k arrays for the kth sub-domain using Lemma 15. It then uses the OPTHIST method on A_k, B_k to produce the kth local optimal B-bucket histogram. Clearly, all m aggregate nodes can run independently

in parallel.

4.4.2 The Partition Phase in the Tuple Model

In the tuple model, the tuples needed to compute $\mathrm{Var}_{\mathcal{W}}[g_i]$ and $\mathrm{E}_{\mathcal{W}}[g_i]$ for each item i are distributed over β tuple chunks. Hence, we rewrite equation (4.6) for computing A, B arrays in the tuple model as follows:

Lemma 16. *The A, B arrays in the tuple model can also be computed as:*

$$A[j] = \sum_{i=1}^{j}(\sum_{\ell=1}^{\beta} \mathrm{Var}_{\mathcal{W},\ell}[g_i] + (\sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i])^2) \;\; B[j] = \sum_{i=1}^{j}\sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i],$$

where $\mathrm{Var}_{\mathcal{W},\ell}[g_i] = \sum_{t\in\tau_\ell} \Pr[t=i](1-\Pr[t=i])$ *and* $\mathrm{E}_{\mathcal{W},\ell}[g_i] = \sum_{t\in\tau_\ell} \Pr[t=i]$.

A similar procedure as that described for the value model could then be applied. The difference is that the ℓth data node processing chunk τ_ℓ emits a key-value pair $(i, (\mathrm{E}_{\mathcal{W},\ell}[g_i], \mathrm{Var}_{\mathcal{W},\ell}[g_i]))$ instead, for *each distinct item i from the union of all possible choices of all tuples in τ_ℓ*. Thus, the kth aggregate node will reconstruct A_k, B_k arrays according to Lemma (16) and then use the OPTHIST method on A_k, B_k arrays to produce the local optimal B-bucket histogram for the kth sub-domain in the partition phase.

4.4.3 Recursive PMERGE and Other Remarks

For RPMERGE, we carry out the partition phase for each model using the method from Section 4.4.1 and Section 4.4.2, respectively. In the merge phase, we can easily invoke multiple independent nodes/processes to run all merging steps in one iteration in parallel. In the following, we denote the distributed and parallel PMERGE and RPMERGE methods as parallel-PMERGE and parallel-RPMERGE, respectively.

4.5 Parallel-PMERGE with Synopsis

A paramount concern in distributed computation is the communication cost. The parallel-PMERGE method may incur high communication cost for large domain size.

This cost is $O(n)$ in the value model. Given a set τ of tuples in a value model database with size $N = |\tau|$, τ is stored in β distributed chunks in a DFS. Each tuple will produce a key-value pair to be emitted by one of the data nodes. In the worst case $N = n$ (one tuple

for each item of the domain), thus $O(n)$ cost. On the other hand, this cost is $O(\beta n)$ in the tuple mode. The worst case is when possible choices from all tuples in every distributed tuple chunk have covered all distinct items from the domain $[n]$.

There are only $O(Bm)$ bytes communicated in the merge phase of parallel-PMERGE for both models, where every aggregate node sends B buckets to a single node for merging. Thus, the communication cost of parallel-PMERGE is dominated by the partition phase.

We present a novel synopsis to address this issue. The key idea is to approximate the A_k, B_k arrays at the kth aggregate node ($k \in [1, m]$) with unbiased estimators $\widehat{A}_k, \widehat{B}_k$ constructed by *either samples or sketches* sent from the data nodes. Since parallel-PMERGE and parallel-RPMERGE share the same partition phase, hence, the analysis above and the synopsis methods below *apply to both methods*.

4.5.1 Sampling Methods for the Value Model

4.5.1.1 The VS Method

One way of interpreting $\mathrm{E}[f_i^2]$ and $\mathrm{E}[f_i]$ is treating each of them as *a count of item* i in the arrays A_k and B_k, respectively. Then $A_k[j]$ and $B_k[j]$ in Lemma 15 can be interpreted as *the rank* of j, i.e., the number of appearance of items from $[s_k, e_k]$ that are less than or equal to j in array A_k, B_k, respectively. Using this view, we show how to construct an estimator $\widehat{B}_k[j]$ with the value model sampling method VS. The construction and results of $\widehat{A}_k[j]$ are similar.

Considering the ℓth data node that processes the ℓth tuple chunk τ_ℓ, we first define $T1(i, \ell) = \mathrm{E}[f_i^2]$ and $T2(i, \ell) = E[f_i]$, respectively, if $(i, f_i) \in \tau_\ell$; otherwise we assign them as 0. We then define $A_{k,\ell}, B_{k,\ell}$ as follows, for every $k \in [1, m]$:

$$A_{k,\ell}[j] = \sum_{i=s_k}^{j} T1(i, \ell), B_{k,\ell}[j] = \sum_{i=s_k}^{j} T2(i, \ell), \text{ for any } j \in [s_k, e_k].$$

Using τ_ℓ, the ℓth data node can easily compute $A_{k,\ell}, B_{k,\ell}$ locally for all k and j values. It is easy to get the following results at *the kth aggregate node* for any $j \in [s_k, e_k]$:

$$A_k[j] = \sum_{\ell=1}^{\beta} A_{k,\ell}[j], B_k[j] = \sum_{\ell=1}^{\beta} B_{k,\ell}[j], \text{ for any } k \in [1, m]. \tag{4.13}$$

We view $B_{k,\ell}[j]$ as the *local rank* of j from τ_ℓ at the ℓth data node. By (4.13), $B_k[j]$ is

simply *the global rank of* j that equals the sum of all local ranks from β nodes. We also let $M_k = \sum_{j=s_k}^{e_k} \mathrm{E}[f_j]$.

For every tuple (i, f_i) from τ_ℓ, data node ℓ unfolds (conceptually) $\mathrm{E}[f_i]$ copies of i, and samples each i independently with probability $p = \min\{\Theta(\sqrt{\beta}/\varepsilon M_k), \Theta(1/\varepsilon^2 M_k)\}$. If a copy of i is sampled, it is added to a sample set $S_{k,\ell}$ where $k = h(i)$, using the hash function in Section 4.4.1. If c_i copies of i are sampled, we add $(i, 1), \ldots, (i, c_i)$ into $S_{k,\ell}$. The pairs of values in $S_{k,\ell}$ are sorted by the item values from the first term, and ties are broken by the second term. Data node ℓ sends $S_{k,\ell}$ to the kth aggregate node for $k \in [1, m]$.

We define the *rank* of a pair (i, x) in $S_{k,\ell}$ as the number of pairs ahead of it in $S_{k,\ell}$, denoted as $r((i, x))$. For any $j \in [s_k, e_k]$ and $\ell \in [1, \beta]$, aggregate node k computes an estimator $\widehat{B}_{k,\ell}[j]$ for the local rank $B_{k,\ell}[j]$ as: $\widehat{B}_{k,\ell}[j] = r((j, c_j))/p + 1/p$, if item j is present in $S_{k,\ell}$.

If an item $j \in [s_k, e_k]$ is not in $S_{k,\ell}$, let y be the predecessor of j in $S_{k,\ell}$ in terms of *item values*, then $\widehat{B}_{k,\ell}[j] = \widehat{B}_{k,\ell}[y] + 1/p$. If no predecessor exists, then $\widehat{B}_{k,\ell}[j] = 0$.

It then estimates the global rank $B_k[j]$ for $j \in [s_k, e_k]$ as:

$$\widehat{B}_k[j] = \sum_{\ell=1}^{\beta} \widehat{B}_{k,\ell}[j]. \tag{4.14}$$

Lemma 17. *$\widehat{B}_k[j]$ in* (4.14) *is an unbiased estimator of $B_k[j]$ and $\mathrm{Var}[\widehat{B}_k[e]]$ is $O((\varepsilon M_k)^2)$.*

The communication cost is $\sum_{\ell,j} p = O(\min\{\sqrt{\beta}/\varepsilon, 1/\varepsilon^2\})$ for $\ell \in [1, \beta]$ and $j \in [s_k, e_k]$ for aggregate node k in the worst case. Hence, the total communication cost in the partition phase of PMERGE with VS is $O(\min\{m\sqrt{\beta}/\varepsilon, m/\varepsilon^2\})$. Note that $\{M_1, \ldots, M_m\}$ can be easily precomputed in $O(m\beta)$ communication cost.

4.5.2 Sketching Methods for the Tuple Model

4.5.2.1 The TS (Tuple Model Sketching) Method

Observe that we can rewrite equations in Lemma 16 to get:

$$A_k[j] = \sum_{\ell=1}^{\beta} \sum_{i=s_k}^{j} \mathrm{Var}_{\mathcal{W},\ell}[g_i] + \sum_{i=s_k}^{j} (\sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i])^2 B_k[j] = \sum_{\ell=1}^{\beta} \sum_{i=s_k}^{j} \mathrm{E}_{\mathcal{W},\ell}[g_i]. \tag{4.15}$$

We can view $\sum_{i=s_k}^{j} \mathrm{Var}_{\mathcal{W},\ell}[g_i]$ and $\sum_{i=s_k}^{j} \mathrm{E}_{\mathcal{W},\ell}[g_i]$ as a local rank of j in a separate

local array computed from τ_ℓ. Similarly, estimation of the global rank, i.e., the first term of $A_k[j]$ and $B_k[j]$ in (4.15), can be addressed by the VS method.

The challenge is to approximate $\sum_{i=s_k}^{j}(\mathrm{E}_{\mathcal{W}}[g_i])^2$, the second term of $A_k[j]$ in (4.15). It is the second frequency moment (F_2) of $\{\mathrm{E}_{\mathcal{W}}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W}}[g_j]\}$. Given that each $\mathrm{E}_{\mathcal{W}}[g_i]$ is a distributed sum and j varies over $[s_k, e_k]$, we actually need a *distributed method* to answer a *dynamic F_2 (energy) range query* approximately on a sub-domain $[s_k, e_k]$.

The key idea is to build AMS sketches [61] for a set of *intervals from a carefully constructed binary decomposition* on each sub-domain locally at every data node.

For a sub-domain $[s_k, e_k]$ at the kth aggregate node, let $M_k'' = \sum_{i=s_k}^{e_k}(\mathrm{E}_{\mathcal{W}}[g_i])^2$. The leaf-level of the binary decomposition partitions $[s_k, e_k]$ into $1/\varepsilon$ intervals, where each interval's F_2 equals $\varepsilon M_k''$. An index-level (recursively) concatenates every two consecutive intervals from the level below to form a new interval (thus, the height of this binary decomposition is $O(\log\lceil\frac{1}{\varepsilon}\rceil)$. Figure 4.2(a) illustrates this idea.

Once the $(\frac{1}{\varepsilon}-1)$ partition boundaries $\{\alpha_{k,1}, \ldots, \alpha_{k,\frac{1}{\varepsilon}-1}\}$ at the leaf-level were found, aggregate node k sends them to all β data nodes. Each data node builds a set of AMS sketches, one for each interval from the binary decomposition (of all levels), over its local data. We denote it as the local *Q-AMS sketch* (Queryable-AMS).

In other words, data node ℓ builds these AMS sketches using $\{\mathrm{E}_{\mathcal{W},\ell}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W},\ell}[g_{e_k}]\}$ as shown in Figure 4.2(b). Then data node ℓ sends its Q-AMS sketch for $[s_k, e_k]$ to the kth aggregate node, which combines β local Q-AMS sketches into a *global Q-AMS* sketch for the kth sub-domain $[s_k, e_k]$, leveraging on the linearly-mergeable property of each individual AMS sketch [36,61]. The global Q-AMS sketch is equivalent to a Q-AMS sketch

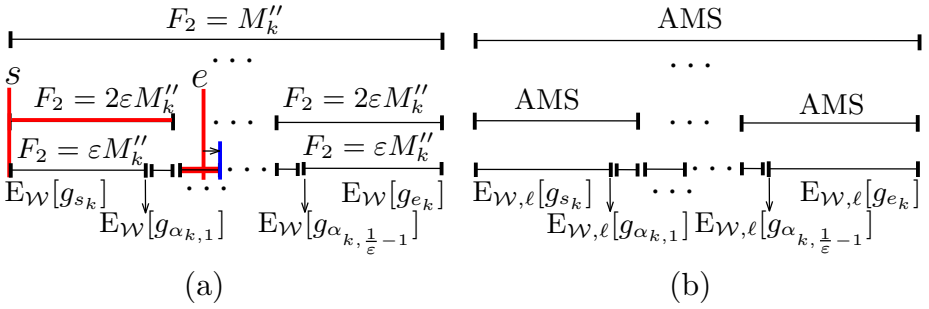


Figure 4.2: Binary decomposition and local Q-AMS. (a) binary decomposition. (b) local Q-AMS.

that is built from $\{\mathrm{E}_{\mathcal{W}}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W}}[g_{e_k}]\}$ directly; recall that $\mathrm{E}_{\mathcal{W}}[g_i] = \sum_{\ell=1}^{\beta} \mathrm{E}_{\mathcal{W},\ell}[g_i]$.

For a query range (s, e), $s_k \leq s < e \leq e_k$, we find the intervals that form the canonical cover of (s, e) in the global Q-AMS sketch, and approximate $(\mathrm{E}_{\mathcal{W}}[g_s])^2 + \cdots + (\mathrm{E}_{\mathcal{W}}[g_e])^2$ by the summation of the F_2 approximations (from the AMS sketches) of these intervals. If (s, e) is not properly aligned with an interval at the leaf-level of an Q-AMS sketch, we snap s and/or e to the nearest interval end point.

The error from the snapping operation in the leaf-level is at most $O(\varepsilon M_k'')$. By the property of the AMS sketch [61], the approximation error of any AMS sketch in the global Q-AMS sketch is at most $O(\varepsilon F_2(I))$, with at least probability $(1 - \delta)$, for an interval I covered by that AMS sketch. Also $F_2(I) \leq M_k''$ for any I in the global Q-AMS sketch. Furthermore, there are at most $O(\log \frac{1}{\varepsilon})$ intervals in a canonical cover since the height of the tree in Q-AMS is $\log\lceil\frac{1}{\varepsilon}\rceil$. Hence, the approximation error for any range F_2 query in the global Q-AMS sketch is $O(\varepsilon M_k'' \log \frac{1}{\varepsilon})$ with probability at least $(1 - \delta)$, for $\varepsilon, \delta \in (0, 1)$ used in the construction of the Q-AMS sketch. Finally, the size of an AMS sketch is $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ [36, 61]. Thus, we can show that:

Lemma 18. *Given the partition boundaries* $\{\alpha_{k,1}, \ldots, \alpha_{k,\frac{1}{\varepsilon}-1}\}$ *for a sub-domain* $[s_k, e_k]$*, for any* s, e *such that* $s_k \leq s < e \leq e_k$*, Q-AMS can approximate* $(\mathrm{E}_{\mathcal{W}}[g_s])^2 + (\mathrm{E}_{\mathcal{W}}[g_{s+1}])^2 + \cdots + (\mathrm{E}_{\mathcal{W}}[g_e])^2$ *within an additive error of* $O(\varepsilon M_k'' \log \frac{1}{\varepsilon})$ *with probability* $\geq (1 - \delta)$ *using space of* $O(\frac{1}{\varepsilon^3} \log \frac{1}{\delta})$.

Each aggregate node needs to send $(\frac{1}{\varepsilon} - 1)$ values per sub-domain to all β data nodes, and there are m sub-domains in total. So the communication cost of this step is $O(m\beta/\varepsilon)$. Then, each data node needs to send out m local Q-AMS sketches, one for each sub-domain. The communication cost of this step is $O(\frac{m\beta}{\varepsilon^3} \log \frac{1}{\delta})$. Hence, the total communication is $O(\frac{m\beta}{\varepsilon^3} \log \frac{1}{\delta})$, which caters for the worst-case analysis.

But the above method and analysis depend on the calculation of the partition boundaries $\{\alpha_{k,1}, \ldots, \alpha_{k,\frac{1}{\varepsilon}-1}\}$ for any sub-domain $[s_k, e_k]$, for $k \in [1, m]$. To calculate this exactly we need $\{\mathrm{E}_{\mathcal{W}}[g_{s_k}], \ldots, \mathrm{E}_{\mathcal{W}}[g_{e_k}]\}$ at the kth aggregate node, which obviously are not available (unless using $O(n\beta)$ total communication for β data nodes for all sub-domains, which defeats our purpose). Fortunately, given that VS can estimate each $B_k[j]$ with an ε error efficiently, each $\mathrm{E}_{\mathcal{W}}[g_i]$ can be estimated as $(\widehat{B}_k[i] - \widehat{B}_k[i-1])$ by (4.15).

4.6 Experiments

We implemented all methods in Java. We test OPTHIST, EF-Histogram, PMERGE, and RPMERGE methods in a centralized environment without parallelism, and parallel-PMERGE and parallel-RPMERGE methods (with and without synopsis) in distributed and parallel settings. The centralized experiments were executed over a Linux machine running a single Intel i7 3.2GHz cpu, with 6GB of memory and 1TB disk space. We then used MapReduce as the distributed and parallel programming framework and tested all methods in a Hadoop cluster with 17 machines (of the above configuration) running Hadoop $1.0.3$. The default HDFS (Hadoop distributed file system) chunk size is 64MB.

4.6.1 Datasets and Setup

We executed our experiments using the WorldCup dataset and the SAMOS dataset. The WorldCup dataset is the access logs of 92 days from the 1998 World Cup servers, composed of 1.35 billion records. Each record consists of client id, file type, and time of access etc. We choose the client id as the item domain, which has a maximum possible domain size of $2,769,184$. We vary the domain size of client ids from $10,000$ up to $1,000,000$. Records in the entire access log are divided into continuous but disjoint groups, in terms of access time. We generate a discrete frequency distribution pdf for items within each grouping interval and assign the pdf to a tuple in the *tuple model*. For the *value model*, we derive a discrete pdf for each client id based on its frequency distribution in the whole log with respect to 13 distinct requested file types and assign the pdf to the tuple with that client id in the value model. The SAMOS dataset is composed of 11.8 million records of various atmospheric measurements from a research vessel and we care about the temperature field, which has a domain size of about 10,000 (by counting two digits after the decimal point of a fraction reading). In a similar way, we form the tuple model and value model data on the SAMOS data.

The default dataset is WorldCup. To accommodate the limited scalability of OPTHIST, we initially vary the value of n from $10,000$ up to $200,000$ and test the effects of different parameters. The default values of parameters are $B = 400$ and $n = 100,000$. For RPMERGE, the recursion depth is $\ell = 2$. We set $m = 16$ and $m = 6$ as the default values for PMERGE and RPMERGE, respectively. We then explore the scalability of our

methods, up to a domain size of $n = 1,000,000$. The running time of all methods is only linearly dependent on N, number of tuples in a database. Hence, we did not show the effect of N; all reported running times are already *start-to-end wall-clock* time.

In each experiment, unless otherwise specified, we vary the value of one parameter, while using the default values of other parameters. The approximation ratios of our approximate methods were calculated with respect to the optimal B-buckets histogram produced by OPTHIST [13, 14].

4.6.2 Centralized Environment

4.6.2.1 Effect of m

Figure 4.3 shows the running time and approximation ratio when we vary m from 4 to 20 on the tuple model datasets. Recall that PMERGE will produce m sub-domains, while RPMERGE will produce m^ℓ sub-domains, in the partition phase. Hence, RPMERGE gives the same number of sub-domains using a (much) smaller m value. For both methods, a larger m value will reduce the size of each sub-domain, hence, reducing the runtime of the OPTHIST method on each sub-domain and the overall cost of the partition phase. But a larger m value increases the cost of the merge phase. As a result, we expect to see a sweet point of the overall running time across all m values. Figure 4.3(a) reflects exactly this trend and the same trend holds on the value model dataset as well. They consistently show that $m = 16$ and $m = 6$ provide the best running time for PMERGE and RPMERGE, respectively. Note that this sweet point can be analytically analyzed, by taking the derivative of the cost function (partition phase + merge phase) with respect to m.

Figure 4.3(b) shows their approximation ratios on the tuple model dataset. The approximation quality of both methods fluctuates slightly with respect to m; but they both produce B-buckets histograms of extremely high quality with approximation ratio very close to 1. The quality is much better than their worst-case theoretical bounds, as indicated by Theorems 8 and 9, respectively.

The results of varying m from the value model are very similar, and have been omitted for brevity. Also, we have investigated the results of varying the recursive depth ℓ from 1 to 3. They consistently show that $\ell = 2$ achieves a nice balance between running time and approximation quality. For brevity, we ommited the detailed results.

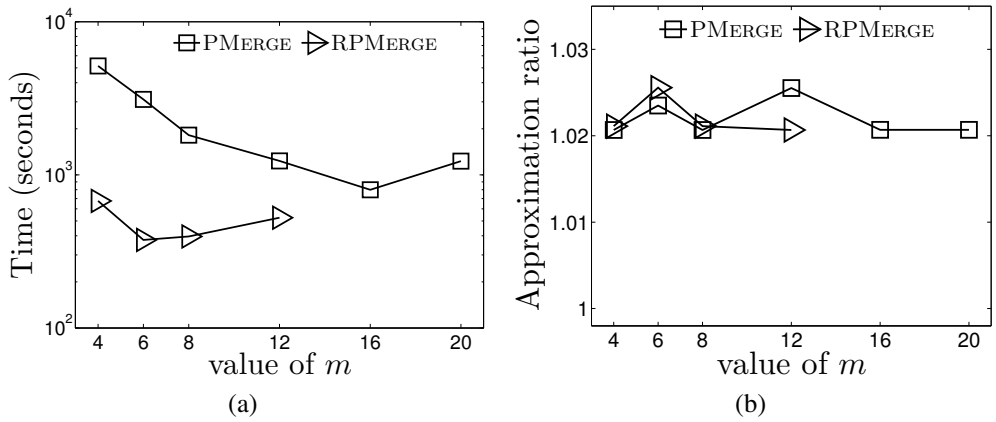


Figure 4.3: Vary m on the tuple model. (a) m vs running time. (b) m vs approximation ratio.

4.6.2.2 Effect of n

Figure 4.4 shows the results with respect to n on both value and tuple models. In both models, the running time of OPTHIST increases quadratically with respect to n. In contrast, both PMERGE and RPMERGE are much more scalable, and have outperformed OPTHIST by at least one to two orders of magnitude in all cases. For example, in Figure 4.4(b), when $n = 100,000$, OPTHIST took nearly 14 hours while RPMERGE took only 861 seconds. RPMERGE further improves the running time of PMERGE by about 2-3 times and is the most efficient method.

Meanwhile, both PMERGE and RPMERGE achieve close to 1 approximation ratios across all n values in Figures 4.4(c) and Figure 4.4(d). The approximation quality gets better (approaching optimal) as n increases on both models.

4.6.2.3 Effect of B

We vary the number of buckets from 100 to 800 in Figure 4.5. Clearly, RPMERGE outperforms OPTHIST by two orders of magnitude in running time in both models, as seen in Figures 4.5(a) and 4.5(b). Figures 4.5(c) and 4.5(d) show the approximation ratios in each model, respectively. The approximation ratio of both PMERGE and RPMERGE slightly increases when B increases on both models. Nevertheless, the quality of both methods are still excellent, remaining very close to the optimal results in all cases.

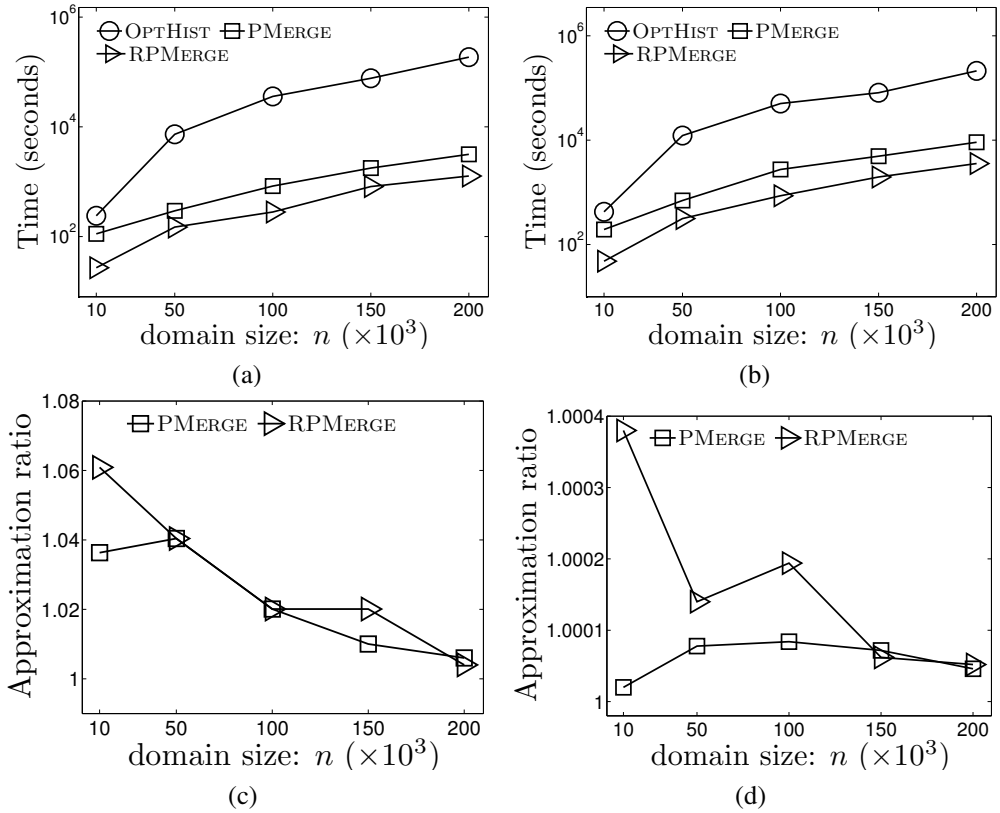


Figure 4.4: Approximation ratio and running time: vary n. (a) Tuple model: running time. (b) Value model: running time. (c) Tuple model: approximation ratio. (d) Value model: approximation ratio.

4.6.2.4 Comparison with the Baseline

Lastly, we compare the running time and approximation ratios of our methods against the baseline EF-Histogram method (with $\varepsilon = 0.1, \varepsilon = 0.05$ and $\varepsilon = 0.01$, respectively) on two datasets. Our methods used their default parameter values on the WorldCup dataset. For the SAMOS dataset, we set $n = 10,000$ and $B = 100$. Clearly, small ε values do help improve the approximation quality of EF-Histogram, as shown in Figure 4.6(b) and Figure 4.6(d). But our methods have provided almost the same approximation quality on both datasets, while offering worst-case bounds in theory as well. Note that the EF-Histogram only provides the $(1+\varepsilon)$ approximation bound with respect to the B-buckets histogram on expected frequencies, but not on the probabilistic histograms.

Meanwhile, the running time of the EF-Histogram increases significantly (it is actually quadratic to the inverse of ε value, i.e., $1/\varepsilon^2$), especially on the much larger WorldCup dataset. In all cases our best centralized method, RPMERGE, has significantly outper-

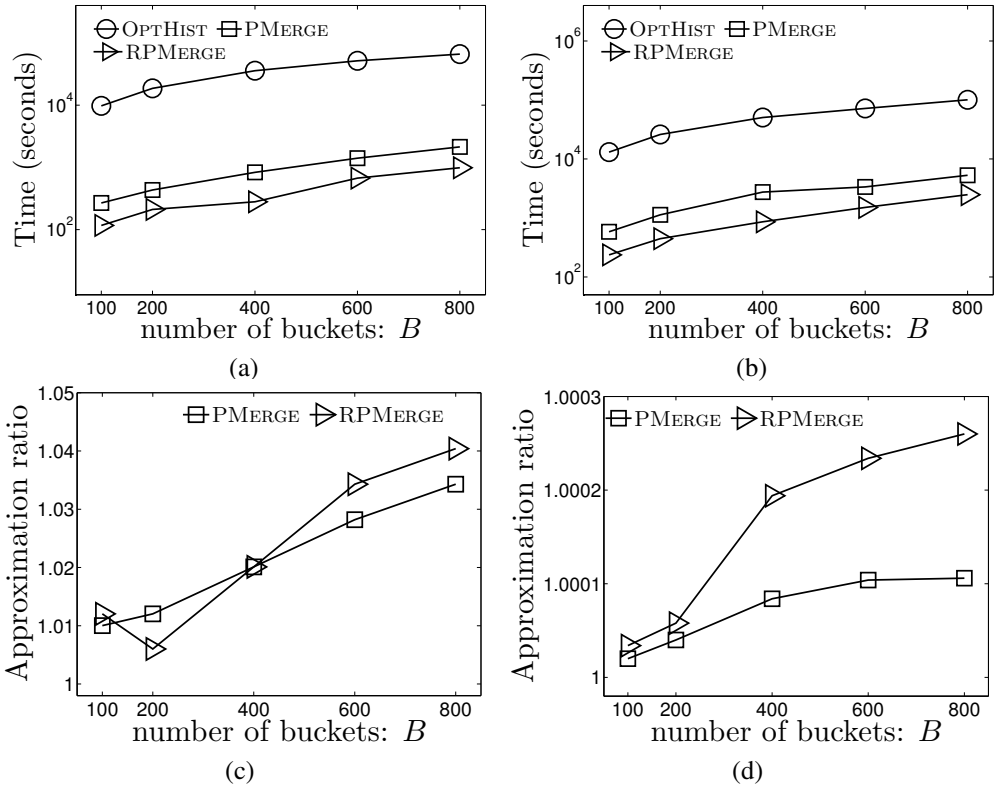


Figure 4.5: Approximation ratio and running time: vary B. (a) Tuple model: running time. (b) Value model: running time. (c) Tuple model: approximation ratio. (d) Value model: approximation ratio.

formed the EF-Histogram, as shown in Figure 4.6(a) and Figure 4.6(c). Furthermore, the distributed and parallel fashion of PMERGE and RPMERGE further improves the efficiency of these methods, as shown next.

4.6.3 Distributed and Parallel Setting

4.6.3.1 Effect of Size of the Cluster

Figure 4.7 shows the running time of different methods when we vary the number of slave nodes in the cluster from 4 to 16. For reference, we have included the running time of centralized PMERGE, and RPMERGE. We can see a (nearly) linear dependency between the running time and the number of slave nodes for both parallel-PMERGE and parallel-RPMERGE methods. The speed up for both methods is not as much as the increasing factor of the number of slave nodes used. The reason is that Hadoop always includes some extra overhead such as job launching and tasks shuffling and IO cost of intermediate HDFS files, which reduces the overall gain from parallelism.

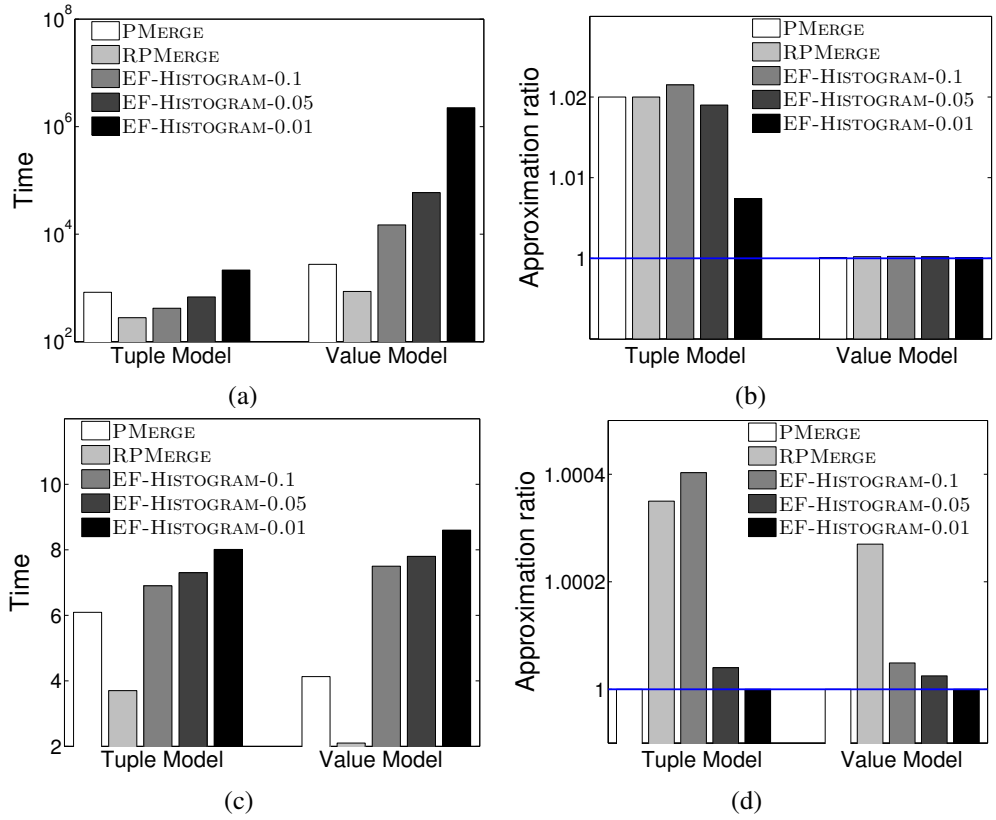


Figure 4.6: Comparison against the baseline method. (a) Running time: WorldCup. (b) Approximation ratio: WorldCup. (c) Running time: SAMOS. (d) Approximation ratio: SAMOS.

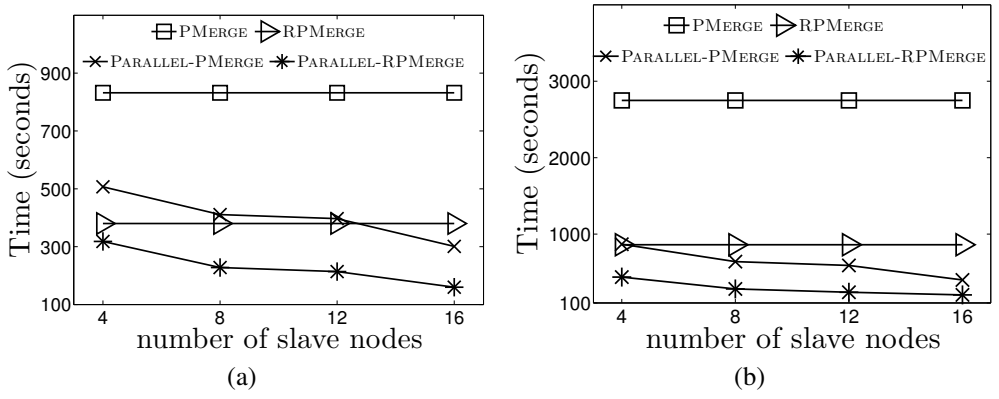


Figure 4.7: Time: vary number of slave nodes. (a) Tuple model. (b) Value model.

4.6.3.2 Scalability

Next, we investigate the scalability of RPMERGE (the best centralized method), parallel-PMERGE, and parallel-RPMERGE on very large probabilistic datasets. We used all 16 slave nodes in the cluster, and varied either the values of n from 200,000 to 1000,000 when $B = 400$, or the values of B from 100 to 800 when $n = 600,000$. We omit OPTHIST and PMERGE methods in this study, since they are too expensive compared to these methods.

Figures 4.8(a) and 4.8(b) show that with recursive merging RPMERGE can even outperform parallel-PMERGE as n increases. But clearly parallel-RPMERGE is the best method and improves the running time of RPMERGE by 8 times on the value model and 4 times on the tuple model when $n = 1000,000$. It becomes an order of magnitude faster than parallel-PMERGE in both models when n increases.

Figures 4.8(c) and 4.8(d) show the running time when we vary B and fix $n = 600,000$. Running time of all methods increase with larger B values. This is because large B values

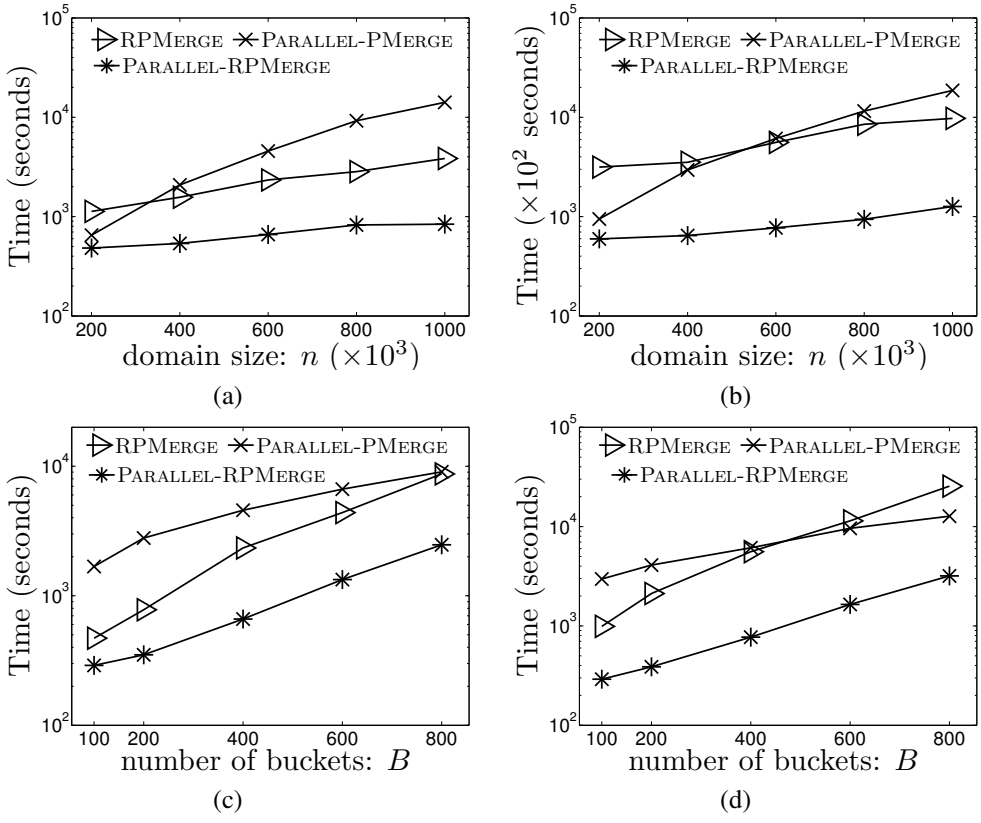


Figure 4.8: Scalability of the parallel approximate methods. (a) Tuple model: vary n. (b) Value model: vary n. (c) Tuple model: vary B. (d) Value model: vary B.

increase the computation cost of the merging step, especially for recursive PMERGE. Nevertheless, parallel-RPMERGE significantly outperforms both parallel-PMERGE and RP-MERGE in all cases on both models.

4.6.4 Distributed and Parallel Synopsis

Lastly, we study the communication saving and approximation quality of parallel-PMERGE and parallel-RPMERGE with synopsis. The default values are $n = 600,000$, $B = 400$ and $\varepsilon = 0.002$ for VS and $\varepsilon = 0.1$ for TS. We have omitted the results for the running time of Parallel-PMERGE and Parallel-RPMERGE with synopsis, since they are very close to that of Parallel-PMERGE and Parallel-RPMERGE, respectively (since the running time of all these methods are dominated by solving the DP instances in the partition and merging phases).

4.6.4.1 Comparing Effects of Synopsis in Both Models

Here, we use parallel-PMERGES (parallel-RPMERGES) to denote a parallel-PMERGE (parallel-RPMERGE) method with a synopsis in either model. In value model, the synopsis is VS; and in tuple model, the synopsis is TS.

Figure 4.9(a) shows that parallel-PMERGES outperforms parallel-PMERGE and parallel-RPMERGE by more than an order of magnitude in communication cost for both models. Parallel-RPMERGES has much higher communication cost than parallel-PMERGES since the sampling cost in the partition phase has increased by an order of m using m^2 sub-domains (when $\ell = 2$). Nevertheless, it still saves about 2-3 times of communication cost compared to that of parallel-PMERGE and parallel-RPMERGE for both models.

Figure 4.9(b) shows that parallel-PMERGES and parallel-RPMERGES have excellent approximation quality on the value model (very close to optimal histograms). They give less optimal approximations in the tuple model, since Q-AMS in the TS method has higher variances in its estimated A, B arrays in the tuple model, compared to the estimations on A, B arrays given by VS in the value model.

The communication cost of all of our synopsis methods are independent of n, whereas the communication cost of both parallel-PMERGE and parallel-RPMERGE are linearly dependent on n, as shown from our analysis in Section 4.5. This means the synopsis methods introduce even more savings when domain size increases.

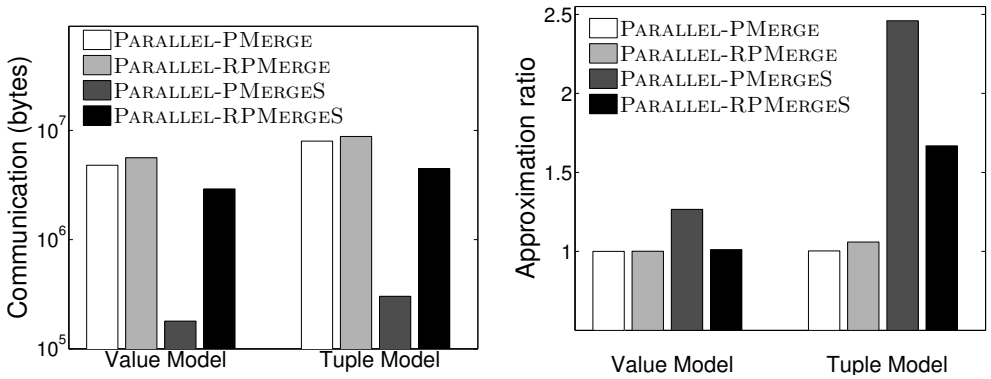


Figure 4.9: Effects of using synopsis. (a) Communication. (b) Approximation ratio.

4.7 Related Work

We have reviewed the most relevant related work in Section 4.2. That said, extensive efforts were devoted to constructing histograms in deterministic data, motivated by the early work in [42, 43, 57, 67]. An extensive survey for histograms on deterministic data is in [41]. There are also numerous efforts on modeling, querying, and mining uncertain data; see [1, 5, 77, 80]. A good histogram for large probabilistic data is very useful for many such operations, e.g, finding frequent items, patterns, and itemsets [1, 5, 80, 90].

However, little was known about histograms over probabilistic data until three recent studies [12–14]. Cormode and Garofalakis have extended the bucket-based histogram and the wavelet histogram to probabilistic data by seeking to minimize the expectation of bucket errors over all possible worlds [13, 14], the detail of which can be found in Section 4.2. Cormode and Deligiannakis then extend the probabilistic histogram definition to allowing a bucket with a pdf representation rather than a single constant value [12]. A main limitation of these studies is the lack of scalability, when the domain size of the probabilistic data increases.

Allowing some approximations in histogram construction is also an important subject on deterministic data, e.g., [35, 78, 79] and many others. One possible choice is to run these methods on expected frequencies of all items, and simply use the output as an approximation to our histogram. But the theoretical approximation bound with respect to the deterministic data (in our case, the expected frequencies of all items) does not carry over to probabilistic histogram definition with respect to n random variables (frequency distributions of every item i). To the best of our knowledge, the $(1 + \varepsilon)$ approximation

from [79] is the best method with theoretical guarantees for histograms over deterministic data (in fact, to the best of our knowledge, other methods are mostly heuristic-based approaches). We did explore this approach as a baseline method in our study.

4.8 Conclusion

In this work, we designed novel approximation methods for constructing optimal histograms on large probabilistic data. Our approximations run much faster and have much better scalability than the state-of-the-art. The quality of the approximate histograms are almost as good as the optimal histograms in practice. We also introduced novel techniques to extend our methods to distributed and parallel settings, which further improve the scalability. Interesting future work include but not limited to how to extend our study to probabilistic histograms with pdf bucket representatives [12] and how to handle histograms of other error metrics.

CHAPTER 5

OTHER WORKS

The theme in this dissertation mainly focuses on efficient and scalable methods for exploring distributed data. Apart from this theme, we have also conducted research for other problems, which include (I) spatial approximate string search; (II) multiple approximate keyword routing (MAKR) in GIS data; (III) ranking large temporal data. The main ideas for these works are summarized as follows.

We have studied the problem of spatial approximate string search [31]. Basically, we investigated range queries augmented with a string similarity search predicate in both Euclidean space and road networks. In Euclidean space, we proposed an approximate solution, the MHR-tree, a customized R-tree which embeds min-wise signatures in each node of the R-tree. The min-wise signature for an index node u keeps a concise representation of the union of q-grams from strings under the sub-tree of u. Therefore, we can analyze the pruning functionality of such signatures based on the set resemblance between the query string and the q-grams from the sub-trees of index nodes. We also discuss how to estimate the selectivity of a spatial approximate string query in Euclidean space. For queries on road networks, we proposed a novel exact method, RSASSOL, which significantly outperforms the baseline algorithm in practice. The RSASSOL combines the q-gram based inverted lists and the reference nodes based pruning.

We also have conducted research to investigate the shortest path search augmented with multiple-approximate-keyword similarity constraint [7]. We proposed two progressive path expansion and refinement algorithms which build up partial candidate paths progressively and refine them until the complete, exact shortest path is guaranteed. We show that the MAKR problem is NP-hard. Thus, we proposed several approximate methods and one of them gives a k approximation ratio for a query set with k keywords. Our approximate methods gives more scalable performance and nice approximation quality in practice.

We investigated aggregate top-k queries on large temporal data in [47]. Objects with top-k highest scores of specified aggregate function over a query range are returned. Our range top-k queries are more flexible and meaningful than instant top-k queries [30]. Particularly, each temporal object is composed by a sequence of line segments to approximate any temporal function in our work. We proposed novel exact methods using B-tree forest and interval tree. We also proposed approximate methods by building indexing structures on carefully constructing break points, where aggregate scores between each consecutive points has equal aggregation scores (except the last one). Therefore, we can snap query range to nearest break-point interval and return its top-k list as an approximate result.

CHAPTER 6

CONCLUSION

In this dissertation, we have studied several emerging problems in exploring and managing distributed data with regards to addressing challenges from uncertainty, large data size, and distributed data sources to support scalable online and offline data exploration tasks.

One observation is that uncertainty becomes very common in modern real-world applications when massive amounts of data are generated. Recent research on probabilistic data management aims to incorporate uncertainty and probabilistic data as "first-class" citizens in the DBMS. Also, handling data uncertainty correctly and efficiently becomes critical for data exploration tasks. To this end, we have revisited the classic problem of threshold monitoring to work on distributed probabilistic data. Correspondingly, we use a probabilistic threshold which contains both score threshold and probability threshold. We proposed exact methods of testing threshold violations for both discrete and continuous probability distribution functions (pdf). We optimized threshold check using sophisticated combinations of tail bounds and combined them with techniques for threshold monitoring on deterministic data to improve communication and computation cost. Further, we proposed several sampling methods to estimate threshold crossing to further improve performance without sacrificing estimation quality. A follow-up problem we have addressed in this dissertation is how to continuous tracking functions of distributed data, instead of only knowing the threshold crossing information in the monitoring problem. We first investigate the online tracking on a chain model, where a tracker is connected to an observer through several relay nodes. Then, we extended online tracking for the max function to "broom" tree model and general-tree model. Our methods can be easily adjusted to track ϕ-quantile, e.g., median, on the broom model. Data summary has received increasing concerns for handling large dataset for many applications, e.g., exploring data distributions, identifying

frequent item sets, etc. Thus, we studied scalable histograms on large probabilistic data leveraging a general partition-merge principle. We further mitigate scalability bottleneck by extending our methods to distributed and parallel settings. We also proposed synopsis based methods to find a trade-off between communication and histogram approximation quality.

In this dissertation, our proposed techniques can be applied to a more general setting for distributed and parallel computation. Interesting future work includes exploring monitoring and tracking more complicate functions and extending our techniques to histograms with other error metrics or pdf bucket representatives.

REFERENCES

[1] AGGARWAL, C. C., LI, Y., WANG, J., AND WANG, J. Frequent pattern mining with uncertain data. In *SIGKDD* (2009).

[2] AGRAWAL, P., BENJELLOUN, O., SARMA, A. D., HAYWORTH, C., NABAR, S., SUGIHARA, T., AND WIDOM, J. Trio: A system for data, uncertainty, and lineage. In *VLDB* (2006).

[3] BABCOCK, B., AND OLSTON, C. Distributed top-k monitoring. In *SIGMOD* (2003), pp. 28–39.

[4] BERESFORD, A. R., AND STAJANO, F. Location privacy in pervasive computing. *IEEE Pervasive Computing 2*, 1 (2003), 46–55.

[5] BERNECKER, T., KRIEGEL, H.-P., RENZ, M., VERHEIN, F., AND ZUEFLE, A. Probabilistic frequent itemset mining in uncertain databases. In *SIGKDD* (2009).

[6] BILLINGSLEY, P. *Probability and measure*. Wiley-Interscience, 1995.

[7] BIN YAO, M. T., AND LI, F. Multi-approximate-keyword routing in gis data. In *SIGSPATIAL* (2011).

[8] CHANDRAMOULI, B., NATH, S., AND ZHOU, W. Supporting distributed feed-following apps over edge devices. *PVLDB 6*, 13 (2013), 1570–1581.

[9] CHANDRAMOULI, B., PHILLIPS, J., AND YANG, J. Value-based notification conditions in large-scale publish/subscribe systems. In *VLDB* (2007), pp. 878–889.

[10] CHENG, R., KALASHNIKOV, D., AND PRABHAKAR, S. Evaluating probabilistic queries over imprecise data. In *SIGMOD* (2003).

[11] CHENG, R., XIA, Y., PRABHAKAR, S., SHAH, R., AND VITTER, J. S. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB* (2004).

[12] CORMODE, G., AND DELIGIANNAKIS, A. Probabilistic histograms for probabilistic data. In *VLDB* (2009).

[13] CORMODE, G., AND GAROFALAKIS, M. Histograms and wavelets on probabilistic data. In *ICDE* (2009).

[14] CORMODE, G., AND GAROFALAKIS, M. Histograms and wavelets on probabilistic data. *IEEE TKDE 22*, 8 (2010), 1142–1157.

[15] CORMODE, G., AND GAROFALAKIS, M. N. Sketching streams through the net: Distributed approximate query tracking. In *VLDB* (2005), pp. 13–24.

[16] CORMODE, G., AND GAROFALAKIS, M. N. Streaming in a connected world: querying and tracking distributed data streams. In *EDBT* (2008).

[17] CORMODE, G., GAROFALAKIS, M. N., MUTHUKRISHNAN, S., AND RASTOGI, R. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD Conference* (2005), pp. 25–36.

[18] CORMODE, G., MUTHUKRISHNAN, S., AND YI, K. Algorithms for distributed functional monitoring. In *SODA* (2008).

[19] CORMODE, G., MUTHUKRISHNAN, S., AND YI, K. Algorithms for distributed functional monitoring. *ACM Transactions on Algorithms 7*, 2 (2011), 21.

[20] CORMODE, G., AND YI, K. Tracking distributed aggregates over time-based sliding windows. In *PODC* (2011), pp. 213–214.

[21] CORMODE, G., AND YI, K. Tracking distributed aggregates over time-based sliding windows. In *SSDBM* (2012), pp. 416–430.

[22] DALVI, N., AND SUCIU, D. Efficient query evaluation on probabilistic databases. In *VLDB* (2004).

[23] DALVI, N., AND SUCIU, D. Efficient query evaluation on probabilistic databases. In *VLDB* (2004).

[24] DESHPANDE, A., GUESTRIN, C., AND MADDEN, S. Using probabilistic models for data management in acquisitional environments. In *CIDR* (2005).

[25] DESHPANDE, A., GUESTRIN, C., MADDEN, S., HELLERSTEIN, J., AND HONG, W. Model-driven data acquisition in sensor networks. In *VLDB* (2004).

[26] DESHPANDE, A., GUESTRIN, C., MADDEN, S., HELLERSTEIN, J. M., AND HONG, W. Model-driven data acquisition in sensor networks. In *VLDB* (2004), pp. 588–599.

[27] DIAO, Y., RIZVI, S., AND FRANKLIN, M. J. Towards an internet-scale xml dissemination service. In *VLDB* (2004), pp. 612–623.

[28] DONG, X., HALEVY, A. Y., AND YU, C. Data integration with uncertainty. In *VLDB* (2007).

[29] FAGIN, R., LOTEM, A., AND NAOR, M. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci. 66*, 4 (2003), 614–656.

[30] FEIFEI LI, K. Y., AND LE, W. Top-k queries on temporal data. *The VLDB Journal-The International Journal on Very Large Data Bases 19*, 5 (2010), 715–733.

[31] FEIFEI LI, BING YAO, M. T., AND HADJIELEFTHERIOU, M. Spatial approximate string search. *TKDE 25*, 6 (2013), 1394–1409.

[32] GAROFALAKIS, M. N., KEREN, D., AND SAMOLADAS, V. Sketch-based geometric monitoring of distributed stream queries. *PVLDB 6*, 10 (2013), 937–948.

[33] GE, T., GRABINER, D., AND ZDONIK, S. B. Monte carlo query processing of uncertain multidimensional array data. In *ICDE* (2011).

[34] GIATRAKOS, N., DELIGIANNAKIS, A., GAROFALAKIS, M. N., SHARFMAN, I., AND SCHUSTER, A. Prediction-based geometric monitoring over distributed data streams. In *SIGMOD* (2012), pp. 265–276.

[35] GIBBONS, P. B., MATIAS, Y., AND POOSALA, V. Fast incremental maintenance of approximate histograms. In *VLDB* (1997).

[36] GRAHAM CORMODE, M. G., AND SACHARIDIS, D. Fast approximate wavelet tracking on streams. In *EDBT* (2006).

[37] GRUENWALD, L., CHOK, H., AND ABOUKHAMIS, M. Using data mining to estimate missing sensor data. In *ICDMW* (2007).

[38] HUA, M., AND PEI, J. Continuously monitoring top-k uncertain data streams: a probabilistic threshold method. *DPD 26*, 1 (2009), 29–65.

[39] HUANG, L., GAROFALAKIS, M., JOSEPH, A. D., AND TAFT, N. Communication-efficient tracking of distributed cumulative triggers. In *ICDCS* (2007).

[40] HUANG, Z., YI, K., AND ZHANG, Q. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *PODS* (2012), pp. 295–306.

[41] IOANNIDIS, Y. E. The history of histograms. In *VLDB* (2003).

[42] IOANNIDIS, Y. E., AND POOSALA, V. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD* (1995).

[43] JAGADISH, H. V., KOUDAS, N., MUTHUKRISHNAN, S., POOSALA, V., SEVCIK, K., AND SUEL, T. Optimal histograms with quality guarantees. In *VLDB* (1998).

[44] JAMPANI, R., XU, F., WU, M., PEREZ, L. L., JERMAINE, C. M., AND HAAS, P. J. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD* (2008).

[45] JAYRAM, T. S., KALE, S., AND VEE, E. Efficient aggregation algorithms for probabilistic data. In *SODA* (2007).

[46] JAYRAM, T. S., MCGREGOR, A., MUTHUKRISHNAN, S., AND VEE, E. Estimating statistical aggregates on probabilistic data streams. In *PODS* (2007).

[47] JEFFREY JESTES, JEFF M. PHILLIPS, F. L., AND TANG, M. Ranking large temporal data. In *VLDB* (2012).

[48] JEYASHANKER, S., KASHYAP, S., RASTOGI, R., AND SHUKLA, P. Efficient constraint monitoring using adaptive thresholds. In *ICDE* (2008).

[49] JIN, C., YI, K., CHEN, L., YU, J. X., AND LIN, X. Sliding-window top-k queries on uncertain streams. In *VLDB* (2008).

[50] KANAGAL, B., AND DESHPANDE, A. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE* (2008).

[51] KASHYAP, S. R., RAMAMIRTHAM, J., RASTOGI, R., AND SHUKLA, P. Efficient constraint monitoring using adaptive thresholds. In *ICDE* (2008), pp. 526–535.

[52] KERALAPURA, R., CORMODE, G., AND RAMAMIRTHAM, J. Communication efficient distributed monitoring of thresholded count. In *SIGMOD* (2006).

[53] KERALAPURA, R., CORMODE, G., AND RAMAMIRTHAM, J. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD Conference* (2006), pp. 289–300.

[54] LI, F., YI, K., AND JESTES, J. Ranking distributed probabilistic data. In *SIGMOD* (2009).

[55] MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst. 30*, 1 (2005), 122–173.

[56] MANJHI, A., SHKAPENYUK, V., DHAMDHERE, K., AND OLSTON, C. Finding (recently) frequent items in distributed data streams. In *ICDE* (2005), pp. 767–778.

[57] MATIAS, Y., VITTER, J. S., AND WANG, M. Wavelet-based histograms for selectivity estimation. In *SIGMOD* (1998).

[58] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[59] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[60] MURTHY, R., IKEDA, R., AND WIDOM, J. Making aggregation work in uncertain and probabilistic databases. *TKDE 23*, 8 (2011), 1261–1273.

[61] NOGA ALON, Y. M., AND SZEGEDY, M. The space complexity of approximating the frequency moments. In *STOC* (1996).

[62] OLSTON, C., JIANG, J., AND WIDOM, J. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD* (2003).

[63] OLSTON, C., LOO, B. T., AND WIDOM, J. Adaptive precision setting for cached approximate values. In *SIGMOD Conference* (2001), pp. 355–366.

[64] PAPAPETROU, O., GAROFALAKIS, M. N., AND DELIGIANNAKIS, A. Sketch-based querying of distributed sliding-window data streams. *PVLDB 5*, 10 (2012), 992–1003.

[65] PEI, J., HUA, M., TAO, Y., AND LIN, X. Query answering techniques on uncertain and probabilistic data: tutorial summary. In *SIGMOD* (2008).

[66] PEREZ, L., ARUMUGAM, S., AND JERMAINE, C. Evaluation of probabilistic threshold queries in MCDB. In *SIGMOD* (2010).

[67] POOSALA, V., IOANNIDIS, Y. E., HAAS, P. J., AND SHEKITA, E. J. Improved histograms for selectivity estimation of range predicates. In *SIGMOD* (1996).

[68] QI, Y., JAIN, R., SINGH, S., AND PRABHAKAR, S. Threshold query optimization for uncertain data. In *SIGMOD* (2010).

[69] ROSS, R., SUBRAHMANIAN, V. S., AND GRANT, J. Aggregate operators in probabilistic databases. *J. ACM 52*, 1 (2005), 54–101.

[70] SAMOS. Shipboard Automated Meteorological and Oceanographic System. http://samos.coaps.fsu.edu.

[71] SARMA, A. D., BENJELLOUN, O., HALEVY, A., NABAR, S., AND WIDOM, J. Representing uncertain data: models, properties, and algorithms. *The VLDB Journal 18*, 5 (2009), 989–1019.

[72] SARMA, A. D., BENJELLOUN, O., HALEVY, A. Y., NABAR, S. U., AND WIDOM, J. Representing uncertain data: models, properties, and algorithms. *VLDBJ 18*, 5 (2009), 989–1019.

[73] SCHILLER, J. H., AND VOISARD, A., Eds. *Location-Based Services*. Morgan Kaufmann, 2004.

[74] SHARFMAN, I., SCHUSTER, A., AND KEREN, D. A geometric approach to monitoring threshold functions over distributed data streams. In *SIGMOD* (2006).

[75] SHARFMAN, I., SCHUSTER, A., AND KEREN, D. A geometric approach to monitoring threshold functions over distributed data streams. In *SIGMOD* (2006), pp. 301–312.

[76] SOLIMAN, M. A., ILYAS, I. F., AND CHANG, K. C.-C. Probabilistic top-k and ranking-aggregate queries. *TODS 33*, 3 (2008), 1–54.

[77] SUCIU, D., OLTEANU, D., RÉ, C., AND KOCH, C. *Probabilistic Databases*. Synthesis Lectures on Data Management. 2011.

[78] SUDIPTO GUHA, K. S., AND WOO, J. Rehist: Relative error histogram construction algorithms. In *VLDB* (2004).

[79] SUDIPTO GUHA, N. K., AND SHIM, K. Approximation and streaming algorithms for histogram construction problems. *TODS 31*, 1 (2006), 396–438.

[80] SUN, L., CHENG, R., CHEUNG, D. W., AND CHENG, J. Mining uncertain data with probabilistic guarantees. In *SIGKDD* (2010).

[81] TRAN, T. T. L., MCGREGOR, A., DIAO, Y., PENG, L., AND LIU, A. Conditioning and aggregating uncertain data streams: Going beyond expectations. *PVLDB 3*, 1 (2010), 1302–1313.

[82] VAPNIK, V., AND CHERVONENKIS, A. On the uniform convergence of relative frequencies of events to their probabilities. *The. of Prob. App. 16* (1971), 264–280.

[83] WANG, S., WANG, G., AND CHEN, J. Distributed frequent items detection on uncertain data. In *ADMA* (2010).

[84] WOO, H., AND MOK, A. K. Real-time monitoring of uncertain data streams using probabilistic similarity. In *RTSS* (2007).

[85] YAO, A. C.-C. Some complexity questions related to distributive computing (preliminary report). In *STOC* (1979), pp. 209–213.

[86] YAO, Y., AND GEHRKE, J. Query processing in sensor networks. In *CIDR* (2003).

[87] YI, K., AND ZHANG, Q. Multi-dimensional online tracking. In *SODA* (2009), pp. 1098–1107.

[88] YI, K., AND ZHANG, Q. Optimal tracking of distributed heavy hitters and quantiles. In *PODS* (2009), pp. 167–174.

[89] YI, K., AND ZHANG, Q. Multidimensional online tracking. *ACM Transactions on Algorithms 8*, 2 (2012), 12.

[90] ZHANG, Q., LI, F., AND YI, K. Finding frequent items in probabilistic data. In *SIGMOD* (2008).